The TEXbook

DONALD E. KNUTH *Stanford University*

Illustrations by
DUANE BIBBY

ADDISON WESLEY PUBLISHING COMPANY

Reading, Massachusetts
Menlo Park, California
Don Mills, Ontario
Wokingham, England
Amsterdam · Mexico City
San Juan · Bogotá · Sydney
Santiago · Singapore · Tokyo

Предисловие

Благосклонный читатель! Представляем вам руководство по системе TEX — новой системе набора, предназначенной для создания красивых книг, и особенно для книг, которые содержат много математики. Приготовив рукопись в формате TEX'а, вы тем самым точно объясните компьютеру, как преобразовать рукопись в страницы, типографское качество которых сравнимо с работой лучших в мире наборщиков; в то же время вам для этого не понадобится делать слишком много сложной работы, а просто напечатать рукопись, как на обычной пишущей машинке. Фактически, общий объем вашей работы, вероятно, будет существенно меньше, чем на машинке, если учесть время, обычно необходимое для проверки напечатанной рукописи, т.к. компьютерные текстовые файлы намного легче изменять и перепечатывать. (Если такие претензии звучат слишком громко, чтобы в них поверить, помните, что они были сделаны автором TEX'а в приподнятом настроении, когда TEX наконец заработал, и в любом случае продолжайте читать).

Это руководство предназначено как для тех, кто никогда не пользовался TEX'ом прежде, так и для опытных пользователей. Другими словами, оно претендует на то, чтобы быть универсальным средством, которое удовлетворяет всех, рискуя не удовлетворить никого. Все, что вам надо знать о TEX'е, где-нибудь в этой книге объяснено, а также объяснено и множество из того, что большинство пользователей не заботит.С одной стороны, если вы готовите простую рукопись, вам вовсе не нужно подробно изучать TEX, но с другой, многое из того, что входит в процесс печати технических книг, очень сложно и, если вы желаете достичь наилучшего эффекта, вам потребуется проникнуть в самые темные уголки TEX'а. Чтобы дать возможность читать это руководство многим типам пользователей, применяется специальный знак для обозначения материала, необходимого только для виртуозов. Когда символ

появляется в начале абзаца, это предупреждает читателя об "опасном повороте", не читайте этот абзац без необходимости. Храбрый и опытный в управлении TEX'ом водитель будет постепенно все далее и далее вступать на эти рискованные территории, но для большинства приложений такие детали не имеют значения.

Все, что вы действительно должны знать, прежде чем читать даль-

ше, это то, как получить на вашем компьютере текстовый файл, используя стандартный редактор. Это руководство объясняет, как должен выглядеть этот файл для того, чтобы TEX понял его, но основные принципы использования компьютера здесь не объясняются. Некоторый предшествующий опыт технической печати будет вполне полезным, если вы планируете делать при помощи TEX'а работу, насыщенную математикой, хотя это и не абсолютно необходимо. TEX будет автоматически выполнять большую часть необходимого форматирования уравнений, но более опытный пользователь способен достичь лучших результатов, т.к. формулы можно печатать многими различными способами.

Некоторые абзацы в этом руководстве настолько экзотичны, что они оценены как

Все, что было сказано о простом знаке опасного поворота, для этих случаев подходит вдвойне. Вам, вероятно, понадобится по меньшей мере месячный опыт работы с TEX'ом, прежде чем вы попытаетесь проникнуть в такие дважды опасные глубины системы; в действительности большинству людей никогда не понадобится знать TEX в таких подробностях, даже если они пользуются им ежедневно. В конце концов, можно водить машину, не зная, как работает мотор. Тем не менее, если вы любознательны, здесь есть для вас широкое поле деятельности (касательно TEX'а, а не автомобиля).

Причиной для таких различных уровней сложности является то, что люди изменяются по мере того, как привыкают к некоторому мощному инструменту. Когда вы впервые попытаетесь воспользоваться TEX'ом, вы обнаружите, что некоторые части его очень просты, в то время как к другим надо привыкнуть. Через день или немного позже, после того, как вы успешно напечатаете несколько страниц, вы станете другим человеком, понятия, которыми вы пользовались с трудом, теперь покажутся естественными, и вы сможете в уме представить конечный результат прежде, чем он будет получен машиной. Но вы, вероятно, захотите достичь большего. После следующей недели работы ваши горизонты раздвинутся шире, и вы еще подрастете и т.д. С годами вы столкнетесь с набором многих видов текста и обнаружите, что ваше обращение с TEX'ом по мере накопления опыта претерпело изменения. Так всегда при освоении мощного инструмента: всегда есть, чему научиться, и всегда есть лучший способ делать то, что делал раньше. На

каждой стадии развития вам понадобится чуть отличающийся вид руководства. Может быть, вы даже захотите написать свое. Обращая внимание на знаки опасного поворота в этой книге, вы сможете лучше сосредоточиться на том уровне, который интересует вас в настоящее время.

Руководства по компьютерным системам обычно скучны для чтения, но имейте в виду: это руководство время от времени содержит ШУТКИ, и вы, читая его, действительно можете получить удовольствие. (Однако, большинство шуток только тогда можно как следует оценить, когда вы поняли техническую суть объясняемого — поэтому советуем читать *внимательно.*)

Другая характерная особенность этого руководства — то, что оно не всегда говорит правду. Когда некоторые понятия TEX'а вводятся неформально, указываются только общие правила. Впоследствии вы обнаружите, что эти правила не строго правильны. Вообще, более поздние главы содержат более достоверную информацию, чем ранние. Автору кажется, что этот умышленный обман в действительности облегчает понимание идей. Поскольку вы понимаете простое, но неточное правило, вам будет нетрудно дополнить это правило исключениями.

Для того, чтобы помочь вам усвоить то, что вы читаете, по всему руководству рассыпаны УПРАЖНЕНИЯ. Вообще предполагается, что каждый читатель попытается выполнить каждое упражнение, за исключением вопросов, которые появляются на участках, помеченных знаком "опасного поворота". Если вы не можете решить задачу, вы всегда можете взглянуть на ответ. Но, пожалуйста, постарайтесь решить ее самостоятельно, тогда вы научитесь большему и быстрее. Даже если вы думаете, что знаете решение, вы для уверенности должны открыть приложение A и проверить его.

Хотя TEX, описанный в этой книге, и похож на первый вариант языка форматирования, новая система отличается от старой буквально тысячами деталей. Оба языка названы TEX, но впредь старый язык будет называться TEX78, и им будут пользоваться все меньше и меньше. Давайте сохраним имя TEX для языка, описанного здесь, т.к. он значительно лучше и больше не будет изменяться.

Я хочу поблагодарить сотни людей, которые помогли мне сформулировать эту "окончательную редакцию" языка TEX, основанную на экспериментах с предварительной версией системы. Моя работа в Стэнфорде поддерживалась National Science Foundation, Office of Naval Research,

IBM Corporation System Development Foundation. Я также хочу поблагодарить Американское математическое общество за поддержку, за организацию группы пользователей TEX'а и за публикацию *TUGboat* newsletter (см. приложение J).

Станфорд, Калифорния — Д. Е. К.
Июнь 1983

Приятно видеть на обложке *'Tis pleasant, sure, to see*
имя рода своего; *one's name in print;*
Книга — это все же книга, *A book's a book,*
хоть вроде нетв ней ничего. *although there's nothing in 't.*
(Перевод Соколова С. Н.)

— BYRON, *English Bards and Scotch Reviewers* (1809)

И тут встает вопрос, *A question arose as to whether*
о том ли мы написали, *we were covering the field*
о чем мы намеривались поведать *that it was intended we should fill*
в этом труде. *with this manual.*

— RICHARD R. DONNELLEY, *Proceedings, United Typothetæ of America* (1897)

Содержание

1	Имя игры	1
2	Издание книги и печатание на машинке	3
3	Управление TEX'ом	9
4	Шрифты	17
5	Группирование	25
6	Работа с TEX'ом	31
7	Как TEX читает то, что вы вводите	47
8	Вводимые символы	55
9	Шрифты	65
10	Размеры	71
11	Боксы	79
12	Клей	87
13	Моды	105
14	Как TEX разбивает абзацы на строки	113
15	Как TEX делает из строк страницы	133
16	Печать математики	155
17	Еще о математике	169
18	Формулы: тонкости набора	195
19	Выделенные уравнения	221
20	Макроопределения	237
21	Создание боксов	263
22	Выравнивание	275
23	Программы вывода	299

24 Обзор вертикальной моды 317
25 Обзор горизонтальной моды 339
26 Обзор математической моды 345
27 Исправление ошибок 353

Приложения

A Ответы на все упражнения 365
B Основные команды 403
C Символьные коды 433
D Маленькие хитрости 439
E Примеры форматов 473
F Таблица шрифтов 499
G Создание боксов из формул 513
H Перенос 523
I Предметный указатель xxx
J Объединение пользователей TEX'а xxx

1

Имя игры

Английское слово technology (технология) происходит от греческого корня, начинающегося буквами $\tau\epsilon\chi\ldots$; и это же самое греческое слово также обозначает искусство. Отсюда имя TEX, которое является заглавной формой от $\tau\epsilon\chi$.

Посвященные произносят χ в слове TEX, как греческое "хи", а не как английское "икс", так что TEX рифмуется со словом "смех". Это тоже самое, как "ch" в таких шотландских словах, как *loch* или как *ach* в немецких словах, это испанское j и русское х. Когда вы перед вашим компьютером произнесете это правильно, терминал может слегка запотеть.

Цель этого упражнения в произношении — напомнить вам, что главной заботой TEX'а являются высококачественные технические рукописи. Его особые акценты — на искусство и технологию, как это и подчеркивается греческим словом. Если вы просто хотите получить достаточно хороший документ, что-нибудь приемлемое и в основном читаемое, но не по-настоящему прекрасное, можно обойтись системой попроще. Цель TEX'а — получить *наивысшее* качество; это требует больше внимания к деталям, но оказывается, что ненамного труднее пройти этот дополнительный путь, зато вы сможете гордиться результатом.

Важно отметить и другую особенность в имени TEX'а: буква "E" в нем слегка ниже других. Это смещение "E" — напоминание, о том, что TEX занимается набором текстов, и это отличает TEX от имен других систем. В действительности, TEX (произнесенный, как *tecks*) — это замечательный *Text EXecutive* процессор, разработанный фирмой Honeywell Information Systems. Так как эти два названия систем произносятся совершенно по-разному, они также по-разному должны и писаться. Правильный способ упоминать TEX в компьютерном файле или в других случаях, которые не позволяют сдвинуть вниз "E" — это печатать `TeX`. Тогда не будет путаницы с похожими именами, а читателям будет подсказка, как правильно произносить.

▸ **Упражнение 1.1**
После того, как вы усвоите материал этой книги, кто вы будете, TEXперт или TEXник?

Зато они дают болезням
модные нелепые имена.

They do certainly give very strange
and new-fangled names to diseases.
— PLATO, *The Republic*, Book 3 (c. 375 B.C.)

Техника! — вопит обиженно Искусство,
И этим словом обзовут усилий и ума плоды
Все те, кто туп, чье сердце пусто,
Кто вял и хил для дела и игры.
(Перевод Соколова С.Н.)

Technique! The very word is like the shriek
Of outraged Art. It is the idiot name
Given to effort by those who are too weak,
Too weary, or too dull to play the game.
— LEONARD BACON, *Sophia Trenton* (1920)
Stanford

2

Издание книги и печатание на машинке

Когда вы впервые начали пользоваться терминалом компьютера, вы, вероятно, должны были привыкать к различиям между цифрой 1 и строчной буквой l. Чтобы сделать следующий шаг к уровню полиграфии, достаточному для издания книг, вам потребуется привыкнуть ко многим подобным вещам; вашим глазам и вашим пальцам понадобится освоить большое количество различий.

Прежде всего, в книгах существуют два вида кавычек, а на пишущей машинке — только один. Даже ваш компьютерный терминал, у которого больше символов, чем у обычной пишущей машинки, вероятно, имеет только неориентированные двойные кавычки ("), потому что ASCII — стандартные коды компьютера — разрабатывались без учета перспективы издания книг. Однако, ваш терминал, вероятно, должен иметь два вида одинарных кавычек ‘ и ’; вторая из них используется как апостроф. Американские клавиатуры обычно содержат левую кавычку, показанную как-нибудь вроде `, и апостроф или правую кавычку, которая выглядит, как ' или ´.

Для получения знака двойной кавычки при помощи TEX'а вы просто печатаете две одинарные кавычки подходящего вида. Например, для получения фразы

“Я понимаю.”

(включая знаки кавычек) вы на вашем компьютере печатаете символы

```
‘‘Я понимаю.’’
```

Стиль печати, имитирующий пишущую машинку, используется в этом руководстве везде, где нужно указать конструкции TEX'а, которые вы должны печатать на вашем терминале, так что символы, реально набираемые на терминале, зрительно отличаются от результатов, которые произвел бы TEX и от комментариев самого руководства. Приведем шрифт, используемый в примерах:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789"#$%&@*+-=,.:;?!
()<>[]{}‘’\|/_^~
```

Примечание переводчика. В переводе этого руководства на русский язык используется русифицированная версия TEX'а, в которой добавлены заглавные и строчные русские буквы:

```
АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЬЫЭЮЯ
абвгдежзиклмнопрстуфхцчшщьыъэюя
```

Если ваш терминал не имеет всех этих символов, не отчаивайтесь; TEX может работать и с теми символами, что у вас есть. Для указания пустого

места в тех случаях, когда важно подчеркнуть, что пустое место напечатано, используется дополнительный символ — *значимый пробел* (*blank space*)

```
␣
```

так что в примере, приведенном выше, вы на самом деле печатали

```
``Я␣понимаю.''
```

Без такого символа вам будет трудно разобраться в невидимых частях некоторых конструкций. Но мы не будем использовать ␣ слишком часто, потому что пробелы обычно достаточно очевидны.

Набор книг значительно отличается от машинописного текста в отношении знаков тире, дефис и знака минус. В хорошей математической книге все эти символы различны. Обычно существуют по меньшей мере четыре разных символа:

- `-` дефис или знак переноса
- – черточка или en-тире (тире длиной примерно в букву n)
- — тире или em-тире (тире длиной примерно в букву m)
- − знак минус

Дефисы используются в составных словах типа “мальчик-с-пальчик” и “N-кратный”. En-тире применяется для указания диапазонов чисел, таких как “страницы 13–34”, “упражнение 1.2.6–34”. Em-тире служит знаком пунктуации в предложениях — это то, что мы обычно называем простым тире. Знак минуса используется в формулах. Добросовестный пользователь TEX’a будет внимательно отличать эти четыре применения. Вот как это делается:

для дефиса надо печатать дефис (`-`)
для en-тире — печатать два дефиса (`--`)
для em-тире — печатать три дефиса (`---`)
для знака минуса — печатать дефис в математическом режиме (`$-$`).

(Математический режим заключается между двумя знаками доллара; это обсуждается позже, и сейчас об этом не надо беспокоиться.)

▸ **Упражнение 2.1**
Объясните, как набрать на TEX’е следующее предложение — Алиса сказала: “Я всегда использую en-тире вместо дефиса, когда указываю в библиографии такие номера страниц, как ‘480–491’. ” .

▸ **Упражнение 2.2**
Что, как вы думаете, случится, если вы напечатаете четыре дефиса подряд?

Если вы внимательно посмотрите на большинство хорошо напечатанных книг, вы обнаружите, что с некоторыми комбинациями букв обращаются, как с одной единицей. Например, это справедливо для f и i в слове

find. Такие комбинации называются *лигатурами* (*ligatures*), и профессиональный наборщик обычно умеет отслеживать такие комбинации букв, как `ff`, `fi`, `fl`, `ffi` и `ffl`. (Причина этого в том, что слова типа find не очень хорошо выглядят в большинстве типов печати, если только для этих рядом стоящих букв не введена какая-нибудь лигатура. Просто удивительно, как часто традиционные лигатуры появляются в английском языке; в других языках важны другие комбинации.)

▸ **Упражнение 2.3**
Придумайте английское слово, содержащее две лигатуры.

Могу вас обрадовать: вы *не* должны сами заботиться о лигатурах — TEX вполне способен обращаться с такими вещами самостоятельно, используя тот же механизм, что и при преобразовании `--` в –. Фактически, TEX будет также следить за комбинациями соседних букв (типа `A` после `V`), которые должны быть подвинуты друг к другу, для того, чтобы они лучше смотрелись; это называется *кернированием* (*kerning*).

Сделаем краткий обзор этой главы. Используя TEX для получения обычного текста, вы печатаете этот текст, как на обычной пишущей машинке, за исключением того, что следует быть внимательными к знакам кавычек, цифре 1 и различным видам дефис/тире. TEX будет автоматически заботиться о других тонкостях, таких как лигатуры и кернирование.

(Вы уверены, что вы должны читать этот абзац? Знак “опасного поворота” означает предупреждение вас о материале, который должен быть пропущен при первом чтении, а может быть, также и при втором. Абзацы, требующие осторожности от читателя, иногда ссылаются на понятия, которые объясняются в более поздних главах.)

Если на вашей клавиатуре нет символа левой кавычки вы можете напечатать `\lq`, за которым следует пробел, если следующий символ является буквой, или `\`, если следующий символ является пробелом. Аналогично, `\rq` дает правую кавычку. Вы понимаете?

```
\lq\lq␣Я␣понимаю.\rq\rq␣
```

В случае, если вам надо напечатать кавычки внутри кавычек, например, простые кавычки перед двойными кавычками, вы не можете просто напечатать `'''` потому что TEX будет интерпретировать это как ”’ (а именно, двойные кавычки, а за ними простые). Если вы уже прочли главу 5, вы могли бы ожидать, что здесь надо будет использовать группирование — печатать что-то вроде `{'}''`. Но оказывается, что это не приведет к желаемому результату, потому что обычно за простой правой кавычкой оставляется меньше места, чем за двойной правой кавычкой. Вы получите ’”, что, конечно, является простой кавычкой перед двойной (если вглядеться достаточно внимательно), но выглядит почти также, как три простые равно отстоящие кавычки. С другой стороны, вы несомненно не захотите печатать `'␣''`, потому что здесь получится слишком большой промежуток, такой

же большой, как между словами — и TEX мог даже начать новую строчку с такого пробела, образуя абзац! Решением является печатать `'\thinspace''`, что даст желаемые '".

▸ **Упражнение 2.4**
Итак, теперь вы знаете, как получить "' и '"; а как получить "` и `"?

▸ **Упражнение 2.5**
Почему, как вы думаете, автор ввел команду `\thinspace` для решения проблемы соседних кавычек, вместо того, чтобы рекомендовать более хитрую конструкцию `'$\,$''` (которая также работает)?

В моде вздор, где мысль убога,
зато тире— и пауз——много.
(Перевод Соколова С.Н.)

In modern Wit all printed Trash, is
Set off with num'rous Breaks——and Dashess—
— JONATHAN SWIFT, On Poetry: A Rapsody (1733)

Некоторые наборщики по-прежнему протестуют
против работы в оффисах,
где внедрены наборные машины.

Some compositors still object
to work in offices
where type-composing machines are introduced.
— WILLIAM STANLEY JEVONS, *Political Economy* (1878)

3

Управление TEX'ом

Количество клавиш на вашей клавиатуре меньше количества символов, которые вы бы хотели задать. Для того, чтобы сделать ограниченную клавиатуру достаточно гибкой, один из символов, которые вы можете вводить, зарезервирован для специального применения и называется *сигнальным символом (escape character)*. Когда вы хотите ввести что-нибудь, что управляет форматом вашей рукописи или что использует клавиатуру необычным образом, вы должны ввести перед указанием того, что вы хотите делать, сигнальный символ.

Примечание. Некоторые терминалы имеют клавишу, помеченную 'ESC', но это *не* то же самое, что ваш сигнальный символ! Это клавиша, которая посылает специальное сообщение операционной системе, поэтому не путайте ее с тем, что называется "сигнальным символом" в данном руководстве.

TEX позволяет использовать для сигнального символа любой символ, но обычно для этих целей принят символ '\' — "обратная косая черта" ("backslash"), так как обратную косую черту удобно печатать, и она редко нужна в обычном тексте. Лучше, если различные пользователи TEX'а работают согласовано, поэтому во всех примерах этого руководства мы будем обозначать сигнальный символ через обратную косую черту.

Непосредственно после '\' (т.е., непосредственно после сигнального символа) вы печатаете команду, говорящую TEX'у, что вы имеете в виду. Такие команды по-английски называются *control sequences*, сокращенно, *cs*. Например, вы могли ввести

```
\input MS
```

что (как вы увидите позже) указывает TEX'у начать чтение файла с именем `MS.tex`; строка символов `\input` — это команда. Приведем другой пример:

```
George P\'olya and Gabor Szeg\"o.
```

TEX преобразует это в" George Pólya and Gabor Szegö.". Тут две команды, `\'` и `\"`; эти команды используются, чтобы поместить над некоторыми буквами знак акцента.

Команды бывают двух видов. Команды первого вида, такие как `\input`, называются *командными словами* (*control word*); они состоят из сигнального символа со следующими за ним одной или несколькими буквами (*letter*), за которыми следует пробел или что-нибудь, кроме букв. (TEX должен знать, где кончается команда, поэтому вы должны поставить пробел после командного слова, если следующий символ является буквой. Например, если вы напечатаете `\inputMS`, TEX естественным образом интерпретирует это, как командное слово из семи букв.) Если вы интересуетесь, что считается "буквой", отвечаем, что TEX так же естественно считает буквами 52 символа `A...Z` и `a...z` . (В русифицированной версии TEX'а буквами также считаются символы `А...Я` и `а...я`.) Десятичные цифры `0...9` буквами *не* считаются, поэтому они не могут появляться в командах первого

вида.

Такие команды как `\'` называются командными символами (*control symbol*); они состоят из сигнального символа со следующей за ним *небуквой*. В этом случае для того, чтобы отделить команду от буквы, которая следует за ним, пробел не нужен, так как команда второго вида всегда содержит после сигнального символа в точности один символ.

▸ **Упражнение 3.1**
Что является командами в '`\I'm \exercise3.1\\!`'?

▸ **Упражнение 3.2**
Мы видим, что если вводить `P\'olya`, то получается Pólya. Можете вы предположить, как должны быть набраны французские слова mathématique и centimètre?

Когда пробел следует после командного слова, он игнорируется TEX'ом, т.е. не рассматривается как "реальный" пробел, принадлежащий набираемой рукописи. Но когда пробел следует после командного символа, то это настоящий пробел.

Теперь встает вопрос, что делать, если вам действительно *надо*, чтобы после командного слова появился пробел? Мы увидим позже, что TEX трактует два и более последовательных пробела как один пробел, поэтому ответ "напечатай два пробела" *не* правилен. Правильным ответом будет напечатать "обязательный пробел", а именно,

```
\␣
```

(сигнальный символ с последующим пробелом); TEX будет трактовать это как пробел, который не игнорируется. Отметим, что обязательный пробел — это команда второго типа, т.е. командный символ, так как за сигнальным символом следует одна небуква (␣). Два последовательных пробела считаются эквивалентными одному пробелу, поэтому дальнейшие пробелы, непосредственно следующие за \␣, будут игнорироваться. Но если вы хотите в рукописи напечатать, скажем, три последовательных пробела, вы можете ввести \␣\␣\␣. Между прочим, наборщиков часто учат ставить по два пробела на концах предложений, но мы увидим позже, что у TEX'а есть собственный способ сделать более широкий пробел в таких случаях. Так что вам не нужно задумываться о количестве пробелов, которые вы вводите.

Непечатающие командные символы типа ⟨return⟩ могут следовать за сигнальным символом, и, в соответствии с правилами, это приводит к особым командам. TEX'у изначально задано трактовать \⟨return⟩ и \⟨tab⟩ так же, как \␣ (обязательный пробел); эти специальные команды, вероятно, не должны переопределяться, потому что, когда вы смотрите на них в файле, вы не можете увидеть различия между ними.

Обычно вам не нужно употреблять "обязательные пробелы", так как команды нечасто нужны в конце слов. Но вот пример, который может

пролить некоторый свет на суть дела: это руководство само напечатано TEX'ом, и в нем, конечно же, часто встречается сама эмблема TEX, в котором буква E сдвинута назад и вниз. Существует специальное командное слово

```
\TeX
```

которое приводит к полудюжине или около того инструкций, которые необходимы для набора эмблемы TEX. Когда нужна такая фраза, как "TEX игнорирует пробелы после командных слов." рукопись содержит ее в таком виде:

```
\TeX\ игнорирует пробелы после командных слов.
```

Отметим дополнительный \, следующий за `\TeX`; это дает обязательный пробел, который необходим, потому что TEX игнорирует пробелы после командных слов. Без этого дополнительного \ в результате было бы

TEXигнорирует пробелы после командных слов.

С другой стороны, мы не можем в любом контексте просто ставить \ после `\TeX`. Например, рассмотрим фразу

```
эмблема    `\TeX'.
```

В этом случае дополнительная обратная косая черта вообще не работает; действительно, если вы введете

```
эмблема    `\TeX\',
```

то получите курьезный результат. Можете вы предположить, что случится? Ответ: `\'` — это команда, обозначающая знак акцента, как в слове `P\'olya` из приведенного выше примера, поэтому в результате получится знак акцента над следующим непустым символом, который в данном случае будет точкой. Другими словами, вы получите точку с акцентом и такой результат:

эмблема `TEX;

Компьютер прекрасно умеет следовать инструкциям, но не читать ваши мысли.

TEX понимает около 900 команд как часть своего встроенного словаря, и все они в данном руководстве где-нибудь объяснены. Но вам не надо заучивать такое множество различных команд, потому что в действительности очень многие из них вам никогда не понадобятся, если, конечно, вы не столкнетесь с необычно сложным документом. Более того, то, что вам надо знать, в действительности попадает в относительно немногие категории, так что может быть усвоено без особых трудностей. Например, многие команды — это просто имена специальных символов, используемых в математических формулах; вы вводите `\pi` — и получите π, из `\Pi` получите Π, из `\aleph` — $\aleph$, из `\infty` — ∞, из `\le` — $\le$, из `\ge` — $\ge$, из `\ne` — $\neq$, из `\oplus` — $\oplus$, из `\otimes` — $\otimes$. Приложение F содержит несколько таблиц таких символов.

В командах не существует встроенного родства между заглавными (uppercase) и строчными (lowercase) буквами. , Например `\pi`, `\Pi`, `\PI` и `\pI` – это четыре разных командных слова.

Около 900 команд, которые только что были упомянуты, в действительности еще не все, потому что их легко определить еще больше. Например, если вы хотите заменить математические символы вашими собственными излюбленными именами, такими, которые вам легче запомнить, вы вольны немедленно сделать это: глава 20 объясняет, как.

Около 300 команд TEX'а называются *примитивами*; это элементарные операции самого низкого уровня, которые не разлагаются на более простые функции. Все остальные определены, в конечном счете, в терминах примитивов. Например, `\input` — примитивная операция, а `\'` и `\"` — нет; они позднее определены в терминах примитивной команды `\accent`.

В рукописях редко используют примитивные команды TEX'а, потому что примитивы ... ну ... слишком *примитивны*. Когда вы пытаетесь заставить TEX делать что-то на нижнем уровне, вы должны вводить множество инструкций; это отнимает время и вносит ошибки. Обычно лучше использовать команды высокого уровня, которые просто объявляют, какие результаты желательны, вместо того, чтобы каждый раз задавать способ достижения каждого результата. Команды высокого уровня надо только однажды определить в терминах примитивов. Например, `\TeX` — это команда, которая означает: "напечатай эмблему TEX"; `\'` — команда, означающая "поставь знак акцента над следующим символом", и обе эти команды могли требовать различных комбинаций примитивов, когда меняется стиль печати. Если была изменена эмблема TEX, автору нужно было бы изменить только одно определение, и изменения автоматически появились бы везде. Напротив, было бы необходимо огромное количество работы, чтобы изменить эту эмблему, если бы она задавалась каждый раз, как последовательность примитивов.

На еще более высоком уровне находятся команды, которые регулируют общий формат документа. Например, в данной книге автор печатал `\exercise` непосредственно перед началом каждого упражнения; эта команда `\exercise` была запрограммирована так, чтобы заставить TEX делать следующие операции:

- вычислить номер упражнения (т.е., '3.2' для второго упражнения главы 3);
- напечатать ▸**УПРАЖНЕНИЕ 3.2** подходящим шрифтом на отдельной строке и с треугольной отметкой в левой части поля;
- оставить маленький дополнительный пробел непосредственно перед этой строкой или, если придется, начать этой строкой новую страницу;
- запретить начало новой страницы сразу после этой строки;
- выключить абзацный отступ на следующей строке.

Очевидно, выгодно избежать ввода всех этих индивидуальных инструкций каждый раз. А после того, как руководство целиком описано в терминах команд высокого уровня, оно может быть напечатано в радикально различных форматах простым изменением примерно дюжины определений.

Как можно отличить примитивы TEX'а от команд, которые были определены как команды высокого уровня? Есть два способа: (1) Алфавитный указатель этого руководства перечисляет все команды, которые обсуждались, и каждый примитив помечен звездочкой. (2) Вы можете вывести значение команды при работе TEX'а. Если вы вводите `\show\cs`, где `\cs` — некоторая команда, TEX ответит ее текущим значением. Например, `\show\input` дает '`> \input=\input.`', потому что `\input` — это примитив. С другой стороны, `\show\thinspace` дает

```
> \thinspace=macro:
->\kern .16667em .
```

Это означает, что `\thinspace` определена как аббревиатура для `\kern .16667em` . Введя `\show\kern`, вы можете убедиться, что `\kern` — это примитив. Результат `\show` появляется на вашем терминале и в протокольном файле, который вы получите после работы TEX'а.

▸ **Упражнение 3.3**
Какая из команд \␣ и \⟨return⟩ является примитивом?

В следующих главах мы будем часто обсуждать "начальный TEX", который состоит из приблизительно 600 основных команд, определенных в приложении B. Эти команды вместе с 300 или около того примитивами обычно присутствуют в начальной стадии, когда TEX приступает к обработке рукописи; вот почему TEX, претендует на знание приблизительно 900 команд в начале работы. Начальный TEX может быть использован для создания документов в гибком формате, который удовлетворяет многие потребности, обходясь шрифтами, которые поставляются системой TEX. Однако, мы должны иметь ввиду, что начальный TEX — это только один из бесчисленных форматов, которые могут быть сконструированы на базе его примитивов; если вам нужен какой-нибудь другой формат, обычно можно так адаптировать TEX, что он будет выполнять все, что было у вас на уме. Лучший путь для изучения — это, вероятно, начать с начального TEX'а, а по мере достижения большего опыта понемногу менять его определения.

Приложение E содержит примеры форматов, которые могут быть добавлены к приложению B для специальных прикладных задач; например, набор определений, подходящих для деловой корреспонденции. Полная спецификация формата, использованного для печати этого руководства, также приводится в приложении E. Так что, если вашей целью является изучение того, как создавать форматы TEX'а, вы, вероятно, захотите изучить приложение E одновременно с освоением приложения B. К тому времени, как вы наберетесь опыта в использовании определений команд, вы, вероятно, разработаете некоторые форматы, которые захотят использовать другие люди, вам тогда надо написать дополнение к этому руководству, объясняющее ваши правила.

Основная цель этих замечаний, адресованных новичку, — убедить, что в TEX'е действительно можно определить нестандартные команды. Когда это руководство говорит, что нечто есть часть "начального TEX'а", это означает, что TEX не настаивает на выполнении этого точно таким же способом: можно изменить правила, заменив одно или более определений в приложении B. Но пока вы не станете опытным TEXническим наборщиком, вы можете спокойно положиться на команды начального TEX'а.

Упражнение 3.4
Сколько может быть различных команд длины 2 (включая сигнальный символ)? Сколько длины 3?

Слоги правят миром.

Syllables govern the world.
— JOHN SELDEN, *Table Talk* (1689)

Я не претендую на то, что я управлял событиями, и честно признаю, что события управляли мной.

I claim not to have controlled events,but confess plainly that events have conntrolled me
— ABRAHAM LINCOLN (1864)

4

Шрифты

Время от времени вам захочется перейти от одного типа печати к другому, например, если вы хотите напечатать **жирно** или *выделить* что-нибудь. TEX имеет дело с наборами, включающими до 256 символов, которые называются шрифтами, а его команды используются, чтобы выбрать конкретный шрифт. Например, вы могли бы задать последние несколько слов первого предложения этого абзаца, используя начальный TEX из приложения B, следующим образом:

```
напечатать  \bf жирно \rm  или \sl выделить \rm что-нибудь.
```

Начальный TEX для изменения шрифтов предусматривает следующие команды:

`\rm` — обычный романский шрифт:	романский (roman)
`\sl` — наклонный романский шрифт:	*наклонный (slanted)*
`\it` — курсив:	*курсив (italic)*
`\tt` — шрифт пишущей машинки:	`пишущая машинка (Typewriter)`
`\bf` — расширенный жирный шрифт:	**Жирный (Bold)**

В начале работы вы получаете романский шрифт (`\rm`), если только вы не указали другое.

Заметим, что два из этих шрифтов имеют "наклон": *наклонный шрифт — это по существу такой же шрифт, как прямой, но его буквы слегка наклонены в сторону, тогда как буквы курсива пишутся по другому.* (Вы сможете, возможно, лучше оценить различие между наклонным шрифтом и курсивом, рассмотрев буквы, *напечатанные ненаклонным курсивом.*) Типографские соглашения в настоящее время находятся в переходном состоянии, потому что новая технология позволила сейчас делать то, что раньше было слишком дорого. Люди ломают голову над вопросом, насколько использовать обретенную вдруг типографскую свободу. Наклонный шрифт был введен в тридцатых годах, но стал широко использоваться в качестве альтернативы общепринятому курсиву в конце семидесятых. Он может быть полезен в математических текстах, т.к. наклонные буквы отличаются от букв курсива, которые используются в математических формулах. Двойное использование курсива для двух различных целей — например, когда утверждения теорем записаны курсивным шрифтом также, как имена переменных в этих теоремах, приводило к путанице, которой теперь с применением наклонного шрифта можно избежать. Вообще, нет согласия в относительных достоинствах наклонного шрифта по сравнению с курсивом, но наклонный шрифт все чаще применяется для названий книг и журналов в библиографиях.

Специальные шрифты подходят для выделения чего-либо, но не для длительного чтения; у вас быстро устали бы глаза, если бы большие разделы этого руководства были целиком напечатаны жирным, наклонным шрифтом или курсивом. Поэтому для большинства печатного материала принят

романский шрифт. Конечно, надоедает ставить `\rm` каждый раз, когда вы хотите вернуться к романскому шрифту, поэтому TEX обеспечивает для этого более легкий способ, используя символ "фигурной скобки" . Вы можете включать шрифты внутри специальных символов `{` и `}`, не затрагивая шрифтов снаружи. Например, фраза, приведенная в начале этой главы, обычно изображается так:

```
напечатать  {\bf жирно}  или {\sl выделить} что-нибудь.
```

Это специальный случай общей идеи "группирования", которую мы обсудим в следующей главе. Лучше всего забыть о первом способе изменения шрифтов, а вместо этого использовать группирование. Тогда ваша TEXовская рукопись будет выглядеть более естественно и вам, вероятно, никогда * не надо будет печатать '`\rm`'.

▸ **Упражнение 4.1**
Объясните, как получить библиографическую ссылку 'Ulrich Dieter, *Journal für die reine und angewandte Mathematik* **201** (1959), 37–70.' [Используйте группирование.]

В предыдущем обсуждении мы слегка затронули важный аспект качества. Взгляните, например, на *курсивные* и *наклонные* слова в этом предложении. Так как курсивный и наклонный шрифты наклонены вправо, они залезают на пробелы, которые отделяют эти слова от следующих за ними прямых слов, в результате промежутки получаются слишком узкими, хотя они и правильны относительно размера букв. Для выравнивания эффективного промежутка TEX позволяет поставить специальную команду `\/` прямо перед переключением обратно на ненаклонные буквы. Когда вы напечатаете

```
{\it курсивные\/} и   {\sl наклонные\/} слова,
```

вы получите *курсивные* и *наклонные* слова, что выглядит лучше. Команда '`\/`' указывает TEX'у добавить к предыдущей букве "*курсивную поправку*", зависящую от этой буквы. Эта поправка в типичном курсивном шрифте для буквы *f* примерно в четыре раза больше, чем для буквы *c*.

Иногда курсивная поправка нежелательна, потому что визуальное расширение обеспечивают другие факторы. Стандартным является применять `\/` непосредственно перед переходом из курсивного или наклонного шрифта в прямой шрифт, кроме случая, когда следующий символ есть точка или запятая. Например

```
{\it курсив\/} для  {\it выделения}.
```

Старые руководства по стилю говорят, что знаки пунктуации после слова должны быть *того же* шрифта, что и *слово,* но курсивная точка с запятой

---

* Ну ладно ..., почти никогда.

часто выглядит плохо, поэтому это соглашение меняется. Когда курсивное слово встречается непосредственно перед точкой с запятой, автор рекомендует печатать `{\it слово\/};`.

▸ **Упражнение 4.2**
Объясните, как напечатать прямое *слово в в середине курсивного предложения.*

У каждой буквы каждого шрифта имеется курсивная поправка, которую вы можете включить, вводя `\/`. Поправка обычно равна нулю в ненаклонных шрифтах печати, но существуют и исключения: чтобы напечатать жирное '**f**' в кавычках, надо сказать `жирное {\bf f\/}`, иначе получается жирное **f**.

▸ **Упражнение 4.3**
Определите команду `\ic` так, чтобы '`\ic c`' помещала курсивную поправку символа *c* в регистр TEX'а `\dimen0`.

Примитивная команда `\nullfont` обозначает шрифт, который не имеет символов. Этот шрифт всегда присутствует, если вы не задали какие-нибудь другие.

Шрифты изменяются по величине так же, как и по форме. Например, шрифт, который вы сейчас читаете, называется "шрифтом в 10 пунктов", потому что отдельные фигуры его конструкций, если их оценивать в печатных единицах, имеют величину в 10 пунктов. (Мы будем изучать пунктовую систему позже, а теперь достаточно обратить внимание, что скобки вокруг этого предложения имеют высоту точно в 10 пунктов, а тире — ширину точно в 10 пунктов . Разделы помеченные знаком "опасного поворота", в этом руководстве напечатаны шрифтом в 9 пунктов, сноски —шрифтом в 8 пунктов, индексы — в 7 пунктов или 6 пунктов, повторные индексы — в 5 пунктов.

Каждый шрифт в рукописи TEX'а связан с командой, например, 10-пунктовый шрифт в этом обзоре вызван `\tenrm`, а соответствующий 9-пунктовый шрифт вызывается `\ninerm`. Наклонные шрифты, которые соответствуют `\tenrm` и `\ninerm`, вызываются `\tensl` и `\ninesl`. Эти команды не встроены в TEX и не являются действительными именами шрифтов; предполагается, что пользователи TEX'а определяют дополнительные имена, когда в рукопись вводятся новые шрифты. Такие команды используются для изменения вида печати.

Когда шрифты различных размеров используются одновременно, TEX выстраивает буквы по их "базовым линиям". Например, если вы печатаете

```
\tenrm меньше \ninerm и меньше
\eightrm и меньше \sevenrm и меньше
\sixrm и меньше \fiverm и меньше \tenrm
```

то в результате будет меньше и меньше и меньше и меньше и меньше. Конечно, к этому ни авторы, ни читатели не привыкли, потому, что наборщики

с традиционными свинцовыми наборами не могли делать таких вещей. Возможно, поэты, которые хотят говорить тихим слабым голосом, будут заставлять выпускать будущие книги, используя частое изменение шрифта, но в настоящее время такие эксперименты любят только редкие любители шрифтов (как автор этого руководства). Не следует увлекаться возможностью переключения шрифта без уважительной причины.

Внимательный читатель здесь должен быть смущен, потому, что в начале этой главы говорилось, что `\rm` — это команда, которая включает романский шрифт, а позднее мы сказали, что это делает `\tenrm`. Правда в том, что работают оба способа. Но обычно сделано так, что `\rm` означает "включи романский шрифт в текущем размере", а `\tenrm` — "включи романский шрифт в 10-пунктовом размере". В начальном TEX'е не предусмотрено ничего, кроме 10-пунктовых шрифтов, поэтому в нем `\rm` будет всегда совпадать с `\tenrm`; но в более сложных форматах TEX'а значение `\rm` в различных частях рукописи будет изменяться. Например, в формате, используемом автором для печати этого руководства, существует команда `\tenpoint`, которая указывает, что `\rm` означает `\tenrm`, `\sl` означает `\tensl`, и т.д., пока `\ninepoint` не изменит определения, так что `\rm` будет означать `\ninerm`, и т.д. Существует другая команда, используемая для набора цитат в конце каждой главы; когда печатаются цитаты, `\rm` и `\sl` временно обозначают, соответственно, 8-пунктовый ненаклонный шрифт "без насечек" ("sans-serif") и *8-пунктовый наклонный шрифт "без насечек"*. Этот механизм постоянно переопределяет аббревиатуры `\rm` и `\sl`, в зависимости от места и освобождает печатающего от необходимости помнить, какой размер или стиль печати используется в текущий момент.

▸ **Упражнение 4.4**

Почему, как вы думаете, автор выбрал названия `\tenpoint`, `\tenrm` и т.д., вместо `\10point` и `\10rm`?

▸ **Упражнение 4.5**

Предположим, что вы напечатали рукопись, используя наклонный шрифт для выделения, а редактор вдруг сказал вам изменить все наклонные слова на курсив. Как это сделать быстро?

Каждый шрифт имеет внешнее имя, которое идентифицирует его относительно других шрифтов в специальной библиотеке. Например, шрифт этого предложения называется '`cmr9`', что является аббревиатурой для "Computer Modern Roman в 9 пунктов." . Для того, чтобы подготовить TEX к использованию этого шрифта, в приложении E появляется команда

```
\font\ninerm=cmr9
```

Обычно, чтобы загрузить информацию о конкретном шрифте в память TEX'а, вы говорите '`\font\cs=`⟨внешнее имя шрифта⟩'; после этого команда `\cs` будет выбирать для печати этот шрифт. Начальный TEX первоначально делает доступными только 16 шрифтов (см. приложение B и приложение F). Но чтобы добрать-

ся до того, что существует в вашей системной библиотеке шрифтов, вы можете использовать `\font`.

Часто есть возможность использовать шрифт нескольких различных размеров, увеличивая или сжимая изображение символов. Каждый шрифт имеет так называемый проектный размер, который отражает размер, обычно присущий этому шрифту по умолчанию; например, проектный размер шрифта `cmr9` — 9 пунктов. Но во многих системах существует также диапазон размеров, в которых вы можете использовать определенный шрифт, уменьшая или увеличивая его размеры. Чтобы загрузить шрифт нужного размера в память TEX'а, вы просто указываете `\font\cs=`⟨внешнее имя шрифта⟩ `at` ⟨желаемый размер⟩. Например, команда

```
\font\magnifiedfiverm=cmr5 at 10pt
```

дает 5-пунктовый Computer Modern Roman шрифт в размере, вдвое большем его обычного. (Предупреждение: перед тем, как использовать эту '`at`', вы должны удостовериться, что электронный наборщик поддерживает шрифт размера, о котором идет речь; TEX допускает любой ⟨желаемый размер⟩, который положителен и меньше, чем 2048 пунктов, но конечный результат будет неправильным, если масштабируемый шрифт отсутствует на устройстве печати).

Каковы различия между `cmr5 at 10pt` и обычным шрифтом в 10 пунктов, `cmr10`? Их много; хорошо сконструированный шрифт в различных размерах пишется по-разному; у букв, чтобы они лучше читались, часто различные пропорции ширины и высоты.

Десятипунктовый шрифт отличается от увеличенного пятипунктового шрифта. Обычно лучше только слегка изменять размеры шрифтов относительно их проектного размера, кроме случаев, когда вы планируете фотографически уменьшить то, что сделал TEX или когда вы добиваетесь необычного эффекта.

Другой способ увеличить шрифт — это указать масштаб. Например, команда

```
\font\magnifiedfiverm=cmr5 scaled 2000
```

это способ увеличить размер шрифта `cmr5` вдвое. Масштаб в TEX'е задается как целое число, равное коэффициенту увеличения, умноженному на 1000. Так, масштаб 1200 означает увеличение в 1.2 раза, и т.д.

▸ **Упражнение 4.6**
Укажите два способа загрузки шрифта `cmr10` в память TEX'а в половину его нормального размера.

Оказалось удобным иметь шрифты с коэффициентами увеличения, составляющими геометрическую прогрессию — что-то вроде хорошо темперированной настройки рояля. Идея в том, чтобы иметь все доступные в истинном размере шрифты также и с увеличением 1.2 и 1.44 (что есть 1.2×1.2), а возможно также с увеличением 1.728 ($= 1.2 \times 1.2 \times 1.2$) и даже выше. Тогда вы сможете увеличивать документ в целом в 1.2 или 1.44 раза и все еще оставаться внутри

набора доступных шрифтов. Начальный TEX содержит аббревиатуры `\magstep0` для масштаба 1000, `\magstep1` для масштаба 1200, `\magstep2` для 1440 и так далее до `\magstep5`. Например, чтобы загрузить шрифт `cmr10` в 1.2×1.2 его нормального размера, вы говорите

```
\font\bigtenrm=cmr10 scaled\magstep2
```

"Это шрифт `cmr10` в нормальном размере (`\magstep0`)."

"Это шрифт `cmr10`, увеличенный в 1.2 (`\magstep1`)."

"Это шрифт `cmr10`, увеличенный в 1.2 (`\magstep2`)."

(Заметим, что маленькое увеличение дает большой эффект). Существует также `\magstephalf`, которое увеличивает в $\sqrt{1.2}$ раза, т.е. посредине между шагами 0 и 1.

Глава 10 объясняет, как применять увеличение к документу в целом вдобавок к увеличению, которое было задано, когда загружался шрифт. Например, если вы загрузили шрифт, который увеличен при помощи `\magstep1` и если вы также задаете

```
\magnification=\magstep2
```

то реальный шрифт для печати будет увеличен с помощью `\magstep3`. Аналогично, если вы загрузили шрифт, измененный командой `\magstephalf` и если вы к тому же говорите

```
\magnification=\magstephalf
```

то напечатанные результаты будут увеличены командой `\magstep1`.

Шрифты — как и лица людей — имеют индивидуальные черты, выдающие их характер.

Type faces—like people's faces—have distinctive features indicating aspects of character.

— MARSHALL LEE, *Bookmaking* (1965)

Это был самый благородный из романского рода.

This was the Noblest Roman of them all.

— WILLIAM SHAKESPEARE, *The Tragedie of Julius Cæsar* (1599)

5

Группирование

Время от времени необходимо рассматривать часть рукописи как одну единицу, поэтому вам надо как-то указать, где эта часть начинается, а где кончается. Для этой цели TEX интерпретирует специальным образом два "символа группирования", (так же, как сигнальный символ), которые трактуются иначе, чем обычные вводимые вами символы. В этом руководстве мы предполагаем, что символами группирования служат `{` и `}`, поскольку именно они используются в начальном TEX'е.

Мы видели пример группирования в предыдущей главе, где упоминалось, что изменение шрифта внутри группы не влияет на шрифт вне ее. Как мы увидим позже, тот же принцип применяется почти ко всему, что определяется внутри группы, например, если вы определите команды внутри некоторой группы, это определение исчезнет, когда группа окончится. Таким образом, можно научить TEX делать что-то необычным способом, временно изменяя соглашения внутри группы, поскольку эти изменения невидимы вне ее. При группировании можно не беспокоиться о путанице в остальной части рукописи, когда необычная конструкция окончилась, а вы забыли восстановить нормальное соглашение. У разработчиков языков программирования есть название для этого аспекта группирования, потому что это — важный общий аспект языков программирования; они его называют "блочной структурой", а определения, которые действуют только внутри группы, называются "локальными" для этой группы.

Можно использовать группирование даже тогда, когда вас не волнует блочная структура, просто чтобы лучше управлять распределением пробелов. Например, давайте рассмотрим еще раз команду `\TeX`, которая дает эмблему TEX. Мы видели в главе 3, что пробел после этой команды будет поглощен, если не ввести '`\TeX\ `'. В то же время было бы ошибкой сказать '`\TeX\`', когда следующий символ не пробел. Однако, во *всех* случаях будет правильным задать простую группу

```
{\TeX}
```

независимо от того, будет или нет следующий символ пробелом, потому что `}` удерживает TEX от поглощения необязательного пробела в `\TeX`. Это может оказаться удобным, когда вы используете текстовой редактор, например, когда некоторые слова замещаются командой. Кроме того, можно ввести

```
\TeX{}
```

используя *пустую* группу: '`{}`' здесь — группа, не имеющая символов, так что она ничего не создает, но удерживает TEX от "съедания" пробелов.

▸ **Упражнение 5.1**
Иногда вы встречаетесь с редким словом типа shelfful, которое лучше выглядит как shelfful, без лигатуры ff . Как вы можете обмануть TEX, чтобы он подумал, что в этом слове нет двух последовательных букв f?

▸ **Упражнение 5.2**
Объясните, как получить три пробела подряд, не используя '\␣'.

TEX также использует группирование для другой, совершенно отличной цели, а именно, чтобы определить, на какую часть вашего текста распространяются некоторые команды. Например, если вы хотите поместить что-то в центр строки, вы можете напечатать

```
\centerline{Эта информация должна быть в центре.}
```

используя команду `\centerline`, определенную в формате начального TEX'а.

Группирование используется в значительной части более сложных инструкций TEX'а, к тому же группа может находиться внутри другой группы, в чем вы можете убедиться, просматривая приложение B. Однако в обычных рукописях сложное группирование, вообще говоря, не является необходимым, поэтому о нем не надо беспокоиться. Не забывайте только закончить каждую группу, которую вы начали, потому что потеря `}` может привести к неприятностям.

Приведем пример двух групп, одна из которых вложена в другую:

```
\centerline{Эта информация должна быть {\it в центре}.}
```

Как и можно было ожидать, TEX выдаст строку, которая содержит курсив:

Эта информация должна быть *в центре*.

Но давайте рассмотрим этот пример более подробно: '`\centerline`' находится вне фигурных скобок, в то время как '`\it`' — внутри. Почему эти два случая различны? И как начинающему научиться запоминать, когда какой из них надо употреблять? Ответ: `\centerline` — это команда, которая применяется только к тому, что следует непосредственно за ней, поэтому вам надо заключить в скобки текст, который вы хотите поместить в центр (если только текст не состоит из простого символа или команды). Например, чтобы поместить слово TEX в центр строки, было бы достаточно напечатать `\centerline\TeX`, но чтобы напечатать фразу "TEX имеет группы", вам нужны скобки: `\centerline{\TeX\ имеет группы}`. С другой стороны, `\it` — команда, которая просто означает "смени текущий шрифт"; она действует, не заглядывая вперед, поэтому влияет на все, что следует далее, или, по крайней мере, может влиять. Скобки окружают `\it` для того, чтобы ограничить локальное изменение шрифта.

Другими словами, два набора скобок в этом примере имеют существенно различные функции: один служит для того, чтобы обращаться с некоторыми словами текста так, как если бы они были простым объектом, в то время как другой обеспечивает локальную блочную структуру.

▸ **Упражнение 5.3**

Как вы думаете, что случится, если вы введете следующее:

```
\centerline{Эта информация должна быть в  {центре}.}
\centerline Так должно быть.
```

▸ **Упражнение 5.4**

А что будет с этим?

```
\centerline{Эта информация должна  быть в  \it центре.}
```

▸ **Упражнение 5.5**

Определите команду `\ital` так, чтобы можно было вводить `\ital{текст}` вместо `{\it текст\/}`. Обсудите за и против `\ital` по сравнению с `\it`.

Последующие главы описывают многие простейшие операции TEX'а, для которых важна локальность группирования. Например, если один из внутренних параметров TEX'а изменился внутри группы, предыдущее состояние этого параметра будет восстановлено, когда группа окончится. Иногда, однако, желательно иметь определение, которое выходит за границы текущей группы. Это можно сделать, если перед определением поставить команду `\global`. Например, TEX хранит номер текущей страницы в регистре `\count0`, а программа, которая выводит страницу, хочет увеличить номер страницы. Программы вывода всегда заключены в группы, поэтому их нельзя нечаянно испортить из остальной части TEX'а, но изменение `\count0` исчезало бы, если бы оно было локальным в программе вывода. Эту проблему решает команда

```
\global\advance\count0 by 1
```

Она увеличивает `\count0` и оставляет его значение таким же после окончания программы вывода. Вообще говоря, `\global` делает определение, которое следует за ним, принадлежащим всем существующим группам, а не только внутренней.

▸ **Упражнение 5.6**

Если вам кажется, что вы поняли локальные и глобальные определения, вот вам маленький тест. Пусть, `\c` означает `\count1=`, `\g` — `\global\count1=`, а `\s` — `\showthe\count1`. Какое значение будет показано?

```
{\c1\s\g2{\s\c3\s\g4\s\c5\s}\s\c6\s}\s
```

Другой способ добиться блочной структуры при помощи TEX'а, — это использовать примитивы `\begingroup` и `\endgroup`. Этими командами легко начать группу внутри одной и закончить ее внутри другой. Текст, который TEX обрабатывает, после расшифровки команд должен иметь правильно вложенные группы, т.е. группы не должны перекрываться. Например,

```
{ \begingroup } \endgroup
```

не разрешается.

Упражнение 5.7

Определите команды `\beginthe`⟨имя блока⟩ и `\endthe`⟨имя блока⟩, которые обеспечат “именованную” блочную структуру. Другими словами, чтобы было допустимо

```
\beginthe{beguine}\beginthe{waltz}\endthe{waltz}\endthe{beguine}
```

а не

```
\beginthe{beguine}\beginthe{waltz}\endthe{beguine}\endthe{waltz}.
```

Я мог прибегнуть к определенным средствам, включая скобки

I have had recourse to varieties of type, and to braces.

—JAMES MUIRHEAD, *The Institutes of Gaius* (1880)

Конфликтная группа — это собрание на несколько часов или дней двенадцати-восемнадцати представительных, ответственных, достоверно нормальных и земных людей.

An encounter group is a gathering, for a few hours or a few days, of twelve or eighteen personable, responsible, certifiably normal and temporarily smelly people.

— JANE HOWARD, *Please Touch* (1970)

6

Запуск TEX'а

Самый лучший способ научиться TEX'у— это начать с ним работать. Поэтому вам самое время сесть за терминал компьютера, попытаться обратиться к TEX'у и посмотреть, что из этого получится. Приведем несколько маленьких, но законченных примеров, которые предлагаются вам для первой попытки.

Берегитесь: эта глава довольно длинная. Почему бы вам не прервать сейчас чтение и не вернуться к нему завтра со свежими силами?

Ладно, давайте предположим, что вы отдохнули и заинтересовались пробным запуском TEX'а. В этой главе приводятся инструкции по его использованию. Во-первых, сделайте вот что. Ступайте в лабораторию, где есть графическое устройство вывода, поскольку вы захотите увидеть результат, который получите — было бы неудобно запускать TEX из удаленного места, где нельзя взять в руки полученный документ. Затем войдите в систему и вызовите TEX. (Можно спросить кого-нибудь, как это сделать на вашем компьютере. Обычно операционная система запрашивает команду, а вы вводите '`TeX`', или '`run tex`', или что-нибудь в этом роде.)

Когда вы справитесь с этим, TEX напечатает:

```
This is TeX, Version 1.0 (preloaded format=plain 83.7.15)
**
```

Символы '`**`' — это запрос TEX'а на имя входного файла.

Теперь введите `\relax` (включая обратную косую черту) и ⟨return⟩ (или что-то другое, что на вашем терминале используется для обозначения "конца строки"). TEX полностью подготовлен к действию и готов читать длинную рукопись, но вы ему говорите, что не стоит беспокоится, так как на этот раз будет простой пример. Действительно, `\relax` является командой, которая означает: "ничего не делай."

Машина напечатает вам другие звездочки. Теперь напечатайте что-нибудь вроде '`Hello?`' и ждите следующих звездочек. Наконец, напечатайте `\end` и посмотрите, что произойдет.

TEX должен ответить '`[1]`' (это значит, что окончилась страница 1 вашего вывода); затем программа остановится, возможно, с сообщением о том, что она создала файл, называемый `texput.dvi`. (TEX для своего вывода использует имя `texput`, если вы не ввели в первой строке что-нибудь получше; а `dvi` обозначает независимость от устройства ("device independent"), поскольку файл `texput.dvi` можно напечатать почти на любом виде печатающих устройств.)

Теперь вам снова потребуется некоторая помощь от доброжелательных местных специалистов. Они подскажут, как получить распечатку с `texput.dvi`. И когда вы увидите распечатку — О, счастливый день! — вы увидите великолепное 'Hello?' и номер страницы '1' . Поздравляем с вашим первым шедевром превосходной печати.

Смысл этой процедуры в том, что вы теперь поняли, как получить что-нибудь от начала до конца. Осталось только проделать то же самое с

каким-нибудь длинным документом. Так, нашим следующим экспериментом будет не вводить текст с терминала, а брать его из файла.

Используйте ваш любимый текстовый редактор для создания файла с именем `story.tex`, который содержит следующие 18 строк текста (ни больше, ни меньше):

```
1  \hrule
2  \vskip 1in
3  \centerline{\bf A SHORT STORY}
4  \vskip 6pt
5  \centerline{\sl by A. U. Thor}
6  \vskip .5cm
7  Once upon a time, in a distant
8    galaxy called \"O\"o\c c,
9  there lived a computer
10 named R.~J. Drofnats.
11
12 Mr.~Drofnats---or ``R. J.,'' as
13 he preferred to be called---
14 was happiest when he was at work
15 typesetting beautiful documents.
16 \vskip 1in
17 \hrule
18 \vfill\eject
```

(Конечно, не печатайте номеров в левой части этих строк, они приведены только для удобства ссылок). Этот пример несколько длинноват и достаточно глуп; но для такого хорошего наборщика, как вы, он не труден, к тому же вы получите некоторый заслуживающий внимания опыт. Поэтому введите его. Для вашей же собственной пользы. И когда вы создаете файл, обдумывайте то, что делаете; этот пример знакомит со многими важными особенностями TEX'а, которые вы можете изучить.

Приведем здесь краткое объяснение того, что вы только что напечатали. Строки 1 и 17 печатают горизонтальную черту (тонкую линию) через всю страницу. Строки 2 и 16 пропускают расстояние в один дюйм: `\vskip` означает "вертикальный пропуск". Такой увеличенный пробел будет отделять горизонтальную черту от остальной части текста. Строки 3 и 5 выводят заголовок и имя автора в центре строки жирным и наклонным шрифтом. Строки 4 и 6 создают дополнительный промежуток между этими и следующими за ними строками (В главе 10 мы будем обсуждать такие единицы измерения, как `6pt` и `.5cm`.)

Основная часть рассказа расположена в строках 7–15 и состоит из двух абзацев . То, что строка 11 пустая, сообщает TEX'у, что строка 10 является концом первого абзаца, а `\vskip` на строке 16 предполагает, что второй абзац кончается на строке 15, потому что вертикальный пропуск не может

появляться внутри абзаца. Между прочим, этот пример оказался очень насыщенным командами TEX'а, но в этом отношении он не типичен. Дело в том, что он специально составлен как учебный. Множество конструкций типа `\vskip` и `\centerline` может появиться в самом начале рукописи, если только вы не используете готовый формат, но это не надолго; в большинстве случаев окажется, что вы печатаете обычный текст с относительно небольшим количеством команд.

Теперь, если вы раньше не вводили текст на компьютере, для вас есть приятный сюрприз: вам не надо заботиться о том, где прервать строку в абзаце (т.е., где оставить правые поля и начать новую строку), потому что TEX сделает это за вас. Подготовленный вами файл может содержать длинные или короткие строки, или и те, и другие — это не имеет значения. Это особенно полезно, когда вносятся изменения, так как вам не надо перепечатывать что-нибудь, за исключением слов, которые изменяются. *Каждое начало строки в файле рукописи — это, по-существу, то же самое, что просто пробел.* Когда TEX прочтет целый абзац — в данном случае строки с 7 по 11 — он будет пытаться разбить текст так, чтобы все строки результата, за исключением последней, были бы приблизительно одинаковой величины, и будет делать перенос слов, если необходимо сохранить разумные постоянные промежутки между словами, но только в крайнем случае.

Строка 8 содержит странное варево:

```
\"O\"o\c c
```

и вы уже знаете, что `\"` обозначает умляут. Командный символ `\c` обозначает "седиль," так что в качестве имени этой далекой галактики вы получите 'Ööç'.

Остальной текст — это просто обзор соглашений по поводу тире и кавычек, которые мы давно обсудили, за исключением новой детали — знаков '`~`' в строках 10 и 12. Они называются *связками или неразрывными пробелами*, поскольку связывают слова вместе. TEX превращает '`~`' в обычный пробел, но не имеет права разрывать в этом месте строку. Хороший наборщик использует неразрывные пробелы между инициалами, как показано в нашем примере. Дальнейшее обсуждение неразрывных пробелов пойдет в главе 14.

Наконец, строка 18 говорит TEX'у `\vfill`, т.е., заполнить оставшуюся часть страницы пробелами; и `\eject` страницу, т.е., послать ее в выходной файл.

Теперь вы готовы к эксперименту номер 2. Запустите TEX снова. Когда машина скажет `**`, вы должны ответить `story`, так как это имя файла, в котором находится ваша входная информация. (Файл можно также назвать его полным именем, `story.tex`, но TEX , если расширение не указано, сам автоматически добавляет расширение `.tex`.)

Вас могло удивить, почему первая подсказка была `**`, а последующие — `*`. Причина этого вот в чем: первое, что вы вводите TEX'у, несколько

отличается от остального: если первый символ вашего ответа на `**` не обратная косая черта, TEX автоматически вставляет `\input`. Таким образом, можно запускать TEX, просто указывая имя входного файла. (Предыдущие системы TEX требовали, чтобы вы начинали работу, вводя `\input story` вместо `story`, и это все еще можно делать; но большинство пользователей TEX'а предпочитают помещать все команды в файл, а не вводить их с терминала, поэтому теперь TEX освобождает их от неприятной обязанности начинать каждый раз с `\input`.) Напомним, что в эксперименте номер 1 вы печатали команду `\relax`, которая начинается с обратной косой черты, поэтому там `\input` не подразумевался.

Существует и другое различие между `**` и `*`: Если первый символ после `**` — это амперсанд (`&`), TEX перед тем, как продолжить работу, заменит свою память на предварительно подготовленный форматный файл. Так, например, если вы пропускаете некоторую версию TEX'а, которая предварительно не загружает начальный формат, то в ответ на `**` можете ввести `&plain \input story` или просто `&plain story`.

Между прочим, многие системы позволяют вызывать TEXпросто указывая одну строку типа `tex story`, не ожидая `**`; Аналогично, в эксперименте номер 1 можно ввести `tex \relax` , а `tex &plain story` загружает начальный формат перед чтением файла `story`. Вам надо проверить, работает ли это на вашем компьютере, или спросить кого-нибудь, есть ли там аналогичное сокращение.

Когда TEX начинает читать ваш файл, он печатает `(story.tex`, возможно, с номером версии для более точной идентификации, в зависимости от вашей операционной системы. Затем он печатает `[1]`, означающее, что сделана страница 1, и `)`, означающее, что файл полностью прочитан.

Затем TeX напечатает `*`, потому что файл не содержит `\end`. Введите `\end`, и тогда должен получится файл `story.dvi` , содержащий оригинал-макет рассказа A.B.Тора. Как и в эксперименте номер 1, можно преобразовать `story.dvi` в распечатку; не мешкайте и сделайте это немедленно. Результат здесь не показан, но вы можете увидеть его, самостоятельно завершив эксперимент. Пожалуйста, сделайте это перед тем, как читать дальше.

▸ **Упражнение 6.1**

Статистика показывает, что только 7.43 из 10 читателей этого руководства действительно напечатали файл `story.tex`, как это было рекомендовано, но что именно эти люди быстрее осваивают TEX. Что мешает вам присоединиться к ним?

▸ **Упражнение 6.2**

Посмотрите внимательно на результат эксперимента номер 2 и сравните его со `story.tex`.Если вы внимательно следовали инструкциям, вы заметите опечатку. Какую и почему она вкралась?

Теперь, вооруженные экспериментом номер 2 , вы знаете, как из файла получить документ. Остальные эксперименты этой главы помогут вам справиться с неизбежными недоразумениями, с которыми вы столкнетесь позже; мы будем умышленно делать то, что заставляет TEX "скрипеть."

Но перед продолжением надо исправить ошибку, замеченную в предыдущей распечатке (см. упражнение 6.2). Строка 13 файла `story.tex` должна быть заменена на

```
he preferred to be called---% error has been fixed!
```

Здесь знак `%` — это деталь начального TEX'а, которую мы прежде не обсуждали. Этот знак ограничивает строку входного файла, не вводя при этом пробела, который TEX обычно вставляет, переходя к следующей строке входного файла. Более того, TEX игнорирует все, что напечатано в файле от `%` до конца строки, поэтому вы можете вставлять в вашу рукопись комментарии, зная, что эти комментарии видимы только вам.

Эксперимент номер 3 усложнит работу TEX'а, потребовав от него помещать рассказ во все более и более узкий столбец. Вот как это делается. После начала программы в ответ на на `**` введите

```
\hsize=4in \input story
```

Это означает: "Расположи рассказ в 4-х дюймовом столбце". Более точно, `\hsize` — это примитив TEX'а, который задает горизонтальный размер абзаца, т.е. ширину каждой нормальной строки результата, а `\input` — это примитив, который указывает TEX'у имя входного файла. Итак, вы задали команды изменить значение `\hsize`, которое определено в начальном TEX'е, а затем обрабатывать `story.tex` в соответствии с этим изменением.

TEX должен ответить, напечатав что-то вроде `(story.tex [1])`, как и прежде, а затем `*`. Теперь введите

```
\hsize=3in \input story
```

а после того, как TEX скажет `(story.tex [2])`, запрашивая следующую команду, введите три дополнительные строки:

```
\hsize=2.5in \input story
\hsize=2in \input story
\end
```

чтобы завершить этот четырехстраничный эксперимент. Не пугайтесь, когда TEX, работая с двухдюймовым размером, несколько раз завопит '`Overfull \hbox`'— это было запланировано в эксперименте номер 3. Просто было невозможно разбить данный абзац на строки шириной в точности два дюйма, если не делать расстояния между словами слишком большими или слишком маленькими. В начальном TEX'е заложены довольно строгие

требования к строкам, которые он создает:

> промежутки между словами не должны быть меньше, чем эти, и
> промежутки между словами не должны быть шире, чем эти.

Если нет способа выполнить эти требования, получается переполненная строка, точнее место ("бокс"), отведенное под строку. А вместе с переполненным боксом получается (1) предупреждающее послание, напечатанное на вашем терминале, и (2) "марашка" (черный прямоугольник), помещенная в документе справа от согрешившего бокса. (Взгляните на страницу 4 результата эксперимента 3; переполненные боксы должны торчать, как бородавки на носу. С другой стороны, страницы 1–3 должны быть безупречны.)

Конечно, вам не нужны такие боксы, поэтому у TEX'а есть несколько способов убрать их; это и будет содержанием эксперимента номер 4. Но сначала давайте поближе посмотрим на результаты эксперимента 3, поскольку TEX, когда его заставляли делать боксы слишком заполненными, сообщил некоторую потенциально ценную информацию. Вы должны научиться читать такие данные:

```
Overfull \hbox (0.98807pt too wide) in paragraph at lines 7--11
\tenrm tant galaxy called []O^^?o^^Xc, there lived|
Overfull \hbox (0.4325pt too wide) in paragraph at lines 7--11
\tenrm a com-puter named R. J. Drof-nats. |
Overfull \hbox (5.32132pt too wide) in paragraph at lines 12--16
\tenrm he pre-ferred to be called---was hap-|
```

Для каждой переполненной строки дается ее положение во входном файле (например, первые два бокса были получены, когда обрабатывался абзац, напечатанный в строках 7–11 файла `story.tex`), а также указывается, насколько не помещается содержимое в боксе (например, на 0.98807 пунктов).

Заметим, что TEX также показывает на экране содержание переполненных боксов в сокращенном виде. Например, последний из них содержит слова 'he preferred to be called—was hap-', напечатанные шрифтом `\tenrm` (романский шрифт в 10 пунктов); а первый — довольно странное воспроизведение 'Ööç', поскольку знаки акцентов появляются в необычных для этого шрифта местах. Вообще, когда вы встречаете в одном из таких сообщений `[]`, это означает либо отступ абзаца, либо некоторую сложную конструкцию; в данном конкретном случае это означает умляут, который расположен над O.

▸ **Упражнение 6.3**
Можете ли вы объяснить '|' , который в этом сообщении появляется после '`lived`'?

▸ **Упражнение 6.4**
Почему перед '|' в '`Drof-nats, |`' стоит пробел?

Вам не надо доставать карандаш и бумагу, чтобы записать полученные сообщения о переполненных боксах, прежде чем они исчезнут из вида, так как TEX всегда ведет "протокол" или "log-файл", в который записывает все интересное, что произошло во время каждого сеанса работы. Например, у вас должен сейчас быть файл с именем `story.log`, содержащий протокол эксперимента 3, а так же файл с именем `texput.log`, содержащий протокол эксперимента 1. (Протокол эксперимента 2 был, вероятно, затерт, когда вы проводили эксперимент 3.) Взгляните теперь на `story.log`. Вы увидите, что сообщения о переполнении боксов сопровождаются не только кратким содержанием боксов, но также некоторыми странного вида данными об h-боксах (hbox), клее (glue), кернах (kern) и тому подобных вещах. Эти данные дают точное описание того, что находится в переполненном боксе. Такие листинги важны для специалистов по TEX'у, если они вызваны, чтобы обнаружить какую-нибудь таинственную ошибку. Вы тоже можете в один прекрасный день захотеть понять внутреннюю кодировку TEX'а.

Сокращенная запись переполненных боксов показывает переносы, которые TEX испробовал перед тем, как прибегнуть к переполнению. Алгоритм переноса, описанный в приложении H, хорош, но не безупречен; например, вы можете увидеть из сообщения в `story.log`, что TEX находит перенос в 'pre-ferred' и даже может перенести 'Drof-nats'. Однако он не нашел переноса в 'galaxy', а иногда проблема переполнения бокса может быть решена, если просто намекнуть TEX'у, как лучше переносить слова. (Мы позже увидим, что для этого есть два способа: либо каждый раз вставлять возможный перенос, как в '`gal\-axy`', либо сказать '`\hyphenation{gal-axy}`' один раз в начале рукописи.)

В нашем примере проблема не в переносе, так как TEX нашел и испробовал все переносы, которые могли помочь. Единственный способ отделаться от переполненных боксов — это изменить допуск (tolerance), т.е., позволить более широкие промежутки между словами. Конечно, допуск, который начальный TEX использует для ширины строк, совершенно не пригоден для 2-х дюймовых колонок; такие узкие колонки просто невозможно получить, не ослабляя ограничений, если специально не подгонять размеры.

TEX присваивает каждой строке числовое значение, называемое "плохость" (badness), которое используется для того, чтобы оценить эстетическое восприятие пробелов между словами. Точные правила вычисления плохости различны для различных шрифтов. Они будут обсуждаться в главе 14, а здесь мы покажем, как работает плохость для романского шрифта начального TEX'а:

Плохость этой строки равна 100.	(очень тесно)
Плохость этой строки равна 12.	(немного тесновато)
Плохость этой строки равна 0.	(хорошо)
Плохость этой строки равна 12.	(чуть свободновато)
Плохость этой строки равна 200.	(свободно)

Плохость этой строки равна 1000. (плохо)
Плохость этой строки равна 5000. (ужасно)

Начальный TEX обычно требует, чтобы ни у одной строки плохость не превышала 200; но в нашем случае выполнение этой задачи было невозможно, поскольку

'tant galaxy called Ööç, there' имеет плохость 1521;
'he preferred to be called—was' имеет плохость 568.

Поэтому теперь мы обратимся к эксперименту номер 4, в котором будем использовать вариации пробелов, более подходящие для узких колонок.

Запустите TEX снова и на этот раз начните с

```
\hsize=2in \tolerance=1600 \input story
```

Это разрешает строки с плохостью до 1600. Ура! На этот раз нет переполненных боксов. (Но зато вы получаете сообщение о *недозаполненном* боксе, поскольку TEX сообщает о всех боксах, плохость которых превышает некоторый порог, называемый `\hbadness`. В начальном TEX'е задано `\hbadness=1000`.) Теперь еще усложним работу TEX'а, попробовав задать

```
\hsize=1.5in \input story
```

(оставляя таким образом допуск равный 1600, но делая колонки еще уже.) Увы, переполненные боксы вернулись; поэтому попробуйте ввести

```
\tolerance=10000 \input story
```

и посмотреть, что получится. TEX рассматривает 10000 как "бесконечный" допуск, позволяя произвольно широкие промежутки; таким образом, допуск, равный 10000, *никогда* не приведет к переполненному боксу, если только не произойдет что-нибудь необычное, вроде непереносимого слова, которое шире, чем сама колонка.

Недозаполненный бокс, который TEX производит в 1.5-дюймовом случае, действительно плох — в таких узких пределах неизбежно время от времени появление широкого пробела. Но попробуйте изменить это, сделав так

```
\raggedright \input story
```

(Это указывает TEX'у не заботиться о сохранении ровных правых полей, но сохраняет единообразные промежутки внутри каждой строки.) И, наконец, в завершении эксперимента 4 введите

```
\hsize=.75in \input story
```

а затем `\end`. Это делает колонку почти невозможно узкой.

Результат этого эксперимента позволяет немного почувствовать задачу разбиения абзаца на приблизительно равные строки. Когда строки относительно широки, TEX почти всегда будет находить хорошее решение. Но в других случаях вы должны будете прийти к некоторому компромиссу, и возможны несколько вариантов. Предположим, вы хотите быть уверенным, что ни у одной строки плохость не превысит 500. Тогда можно установить `\tolerance` равным какому-нибудь большому числу, а `\hbadness=500`; TEX не получит переполненных боксов, но предупредит о недозаполненных. Или можно установить `\tolerance=500`; тогда TEX может получить переполненные боксы. Вторая альтернатива лучше, поскольку тогда вы можете посмотреть на переполненные боксы, чтобы понять, насколько они не помещаются, и становится графически ясно, какое лекарство возможно. С другой стороны, если вам некогда рассматривать плохие промежутки — вы хотите только знать, насколько они плохи — тогда лучше первая альтернатива, хотя она может потребовать дополнительного машинного времени.

▸ **Упражнение 6.5**
Когда задано `\raggedright` (неровный правый край), плохость отражает величину промежутка на правом поле, а не пробелы между словами. Придумайте эксперимент, в котором можно было бы легко определить, какую плохость назначит TEX каждой строке `story`, когда он печатается с неровным правым краем и в 1.5-дюймовой колонке.

Параметр, называемый `\hfuzz`, позволяет игнорировать незначительное переполнение боксов. Например, если вы говорите `\hfuzz=1pt`, бокс, чтобы его признали ошибочным, должен высунуться больше, чем на один пункт. Начальный TEX устанавливает `\hfuzz=0.1pt`.

▸ **Упражнение 6.6**
Изучение результатов эксперимента 4, особенно страницы 3, показывает, что при узких колонках лучше позволить пробелам проявляться до и после тире, когда уже растянуты другие промежутки этой строки. Определите макрокоманду `\dash`, которая делает это.

Вас предупреждали, что это длинная глава. Но соберитесь с духом: осталось провести еще один эксперимент, и вы будете знать о TEX'е достаточно для того, чтобы в дальнейшем бесстрашно самостоятельно его запускать. Единственное, что вы еще упустили — это некоторую информацию о том, как бороться с сообщениями об ошибках — т.е., не только с предупреждениями типа переполнения боксов, но и со случаями, когда TEX останавливается и спрашивает вас, что делать дальше.

Сообщения об ошибках могут испугать, если вы к ним не готовы, но при правильном отношении они могут быть забавными. Только запомните, что вам не следует обижаться на мнение компьютера и что никто не будет использовать ошибки против вас. Тогда окажется, что работа TEX'а не пугает вас, а обогащает творческим опытом.

Первым шагом в эксперименте номер 5 будет внести две преднаме-

ренные ошибки в файл `story.tex`. Замените строку 3 на

```
\centerline{\bf A SHORT \ERROR STORY}
```

а в строке 2 замените '`\vskip`' на '`\vship`'.

Теперь запустите TEX снова, но вместо '`story`' напечатайте '`sorry`'. Компьютер должен ответить, что невозможно найти файл `sorry.tex` и попросить вас попробовать еще раз. На этот раз нажмите ⟨return⟩; вы увидите, что вам следовало задать имя реального файла. Так что напечатайте '`story`' и ждите, пока TEX найдет один из *faux pas* в этом файле.

Ну вот, машина скоро остановится,* напечатав что-нибудь вроде:

```
! Undefined control sequence.
l.2 \vship
           1in
?
```

TEX начинает это сообщение об ошибках с '`!`' и, печатая две строки контекста, показывает, что было прочитано во время обнаружения ошибки. Верхняя часть этой пары (в данном случае '`\vship`') показывает то, что TEX просмотрел до сих пор (`l.2`—это номер файла и номер строки), нижняя часть (в данном случае '`1in`') показывает остаток входной строки.

Знак '`?`', который появляется после контекста, означает, что TEX хочет получить совет, что делать дальше. Если вы перед этим никогда не видели сообщения об ошибках, или если вы забыли, какой ответ ожидается, вы можете ввести '`?`' (двигайтесь вперед и попробуйте сделать это!); TEX откликнется следующим образом:

```
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
```

Это перечень возможностей, а именно:

1. ⟨return⟩: продолжай работу, попытавшись, насколько это можно, исправить ошибку.
2. `S`: продолжай работу, не спрашивая указаний, даже если встретятся ошибки. Последующие сообщения об ошибках промелькнут на вашем терминале, возможно, быстрее, чем вы успеете прочитать их, и появятся в протокольном файле, где на досуге их можно тщательно изучить. Таким образом, `S`— это то же самое, что отвечать ⟨return⟩ на каждое сообщение.

---

* Некоторые версии TEX'а не разрешают взаимодействия. В таких случаях все, что вы можете сделать — это взглянуть на сообщения об ошибках в протоколе, где они появятся вместе со "вспомогательной" информацией.

3. `R`: почти то же самое, что `S`, но сильнее, поскольку приказывает TEX'у не останавливаться ни по какой причине, даже если не найдено имя файла.
4. `Q`: похоже на `R`, но еще сильнее, поскольку приказывает TEX'у не только продолжать работу без остановки, но также подавляет весь дальнейший вывод на терминал. Это быстрый, но несколько безрассудный способ работы (предназначенный для работы TEX'а без присутствия человека).
5. `I`: вставь текст, следующий за `I`. TEX прочтет его, а затем продолжит чтение входной строки. Строки, вставленные таким способом, не должны оканчиваться пробелом.
6. Небольшое число (меньше 100): удали это количество символов и команд из оставшейся части входной информации и остановись снова, чтобы дать еще один шанс посмотреть на то, что получилось.
7. `H`: помоги! Именно это, вы должны сделать сейчас и делать всякий раз, когда сталкиваетесь с сообщением об ошибке, которое вы пока еще не видели. У TEX'а есть два сообщения, встроенных для того, чтобы ошибку мог понять каждый: одно формальное и одно неформальное. Формальное сообщение печатается первым (например, '`! Undefined control sequence.`'); неформальное печатается, если вы, напечатав '`H`', запрашиваете дополнительную помощь, а также оно появляется в протокольном файле, если вы записываете туда сообщения об ошибках. Неформальное сообщение пытается дополнить формальное пояснением того, в чем, по мнению TEX'а, заключается неприятность, и часто советует стратегию для возмещения потерь.
8. `X`: кончай работу, прекратив обработку вашего файла и завершив протокольный файл. `DVI`—файл будет содержать все предыдущие страницы. Текущая (незавершенная) страница выведена не будет.
9. `E`: редактируй — похоже на `X`, но дополнительно подготавливает компьютер к редактированию файла, который TEX в данный момент читает, в текущей позиции, так что вы можете со всеми удобствами сделать изменения перед следующим запуском TEX'а.

После того, как вы введете `H` (или `h`, что тоже работает), вы получите сообщение, которое пытается объяснить вам, что команде, только что прочитанной TEX'ом (т.е., `\vship`), никогда не присваивалось никакого значения, и что вы должны либо вставить правильную команду, либо продолжить работу, как если бы неприятное сообщение и не появлялось.

Поэтому в нашем случае лучше всего ввести

```
I\vskip
```

(и ⟨return⟩), без пробела после '`I`'; это заменяет `\vship` на `\vskip`. (Сделайте это.)

Если бы вы вместо того, чтобы вставить что-либо, просто напечатали ⟨return⟩, TEX пошел бы дальше и прочел '`1in`', что рассматривалось бы как часть формируемого абзаца. Или вы могли напечатать `3`, что удаляет '`1in`' из входного файла. Или для того, чтобы исправить опечатку в файле, можно было напечатать `X` или `E`. Но обычно лучше постараться за один проход TEX'а обнаружить возможно большее число ошибок, так как это увеличивает вашу производительность и в то же время уменьшает затраты машинного времени. Глава 27 более подробно обсуждает искусство управления TEX'ом при ошибочном тексте.

▸ **Упражнение 6.7**
Что случится, если после ошибки `\vship` вы напечатаете '`5`'?

Можно управлять уровнем взаимодействия, задавая команды во входном файле так же, как и с терминала. Примитивы TEX'а `\scrollmode`, `\nonstopmode` и `\batchmode` аналогичны, соответственно, вводу `S`, `R` или `Q` в ответ на сообщение об ошибке, а `\errorstopmode` возвращает вас на обычный уровень взаимодействия. (Такие изменения глобальны независимо от того, появляются они внутри или вне группы.) Более того, многие версии обеспечивают способ прерывания TEX'а во время выполнения; такое прерывание указывает программе вернуться в `\errorstopmode`, после чего она останавливается и ждет дальнейших инструкций.

Что случится дальше в эксперименте номер 5? TEX запнется на следующей ошибке, которую мы внесли в файл. На этот раз, однако, сообщение об ошибке более сложное, поскольку контекст появляется на шести строках вместо двух:

```
! Undefined control sequence.
<argument> \bf A SHORT \ERROR
                                STORY
\centerline #1->\line {\hss #1
                                 \hss }
l.3 \centerline{\bf A SHORT \ERROR STORY}

?
```

Вы получаете такое многострочное сообщение об ошибке, если ошибка обнаружена в момент, когда TEX обрабатывает некоторые команды высокого уровня — в данном случае, он пытался выполнить `\centerline`, которая не является примитивной операцией (она определена в начальном TEX'е). Сначала такое сообщение об ошибке покажется вам полной чепухой, потому что многое из того, что появилось на экране —это команды нижнего уровня, которые вы никогда не писали. Но вы можете выйти из шока, разобравшись в способе, которым действует TEX.

Прежде всего отметим, что дополнительная информация всегда состоит из пар строк. Как и раньше, верхняя строка показывает то, что TEX только что прочитал ('`\bf A SHORT \ERROR`'), затем идет то, что он еще

намеревается прочесть (‘STORY’). Следующая пара строк показывает контекст первых двух; она указывает, что делал TEX непосредственно перед тем, как начал читать остальные строки. В данном случае мы видим, что TEX только что прочел ‘#1’, — специальный код, который говорит машине “читай первый аргумент, который запрашивается текущей командой”, т.е., “сейчас читай материал, который \centerline предполагает поместить в центр строки”. Определение в приложении B говорит, что \centerline, когда применяется к некоторому тексту, предполагает вставку этого текста вместо ‘#1’ в ‘\line{\hss#1\hss}’. Так что TEX находится внутри этой расшифровки \centerline, и к тому же внутри текста, который должен быть помещен в центр строки.

Нижняя строка показывает, как далеко к этому времени TEX продвинулся в файле story. (В нашем примере нижняя строка пустая; то, что кажется нижней строкой, в действительности является первой из двух строк контекста, а это указывает, что TEX прочел все, включая } в третьей строке файла.) Таким образом, данное сообщение об ошибке дает нам первое впечатление о том, как TEX выполнял свою работу. Сначала он увидел \centerline в начале строки 3. Потом посмотрел определение \centerline и заметил, что \centerline имеет “аргумент”, т.е., что \centerline применяется к символу команды или к группе, которая следует за ним. Поэтому TEX прочел и подготовил ‘\bf A SHORT \ERROR STORY’ в качестве аргумента для \centerline. Потом он начал читать расшифровку, как она определена в приложении B. Когда он достиг #1, то начал читать аргумент, который заготовил. А когда он добрался то \ERROR, то пожаловался на неопределенную команду.

▸ **Упражнение 6.8**

Почему TEX не пожаловался на то, что \ERROR не определена, когда впервые натолкнулся на \ERROR, т.е., перед тем как прочел ‘STORY}’ на строке 3?

Когда вы получаете такое многострочное сообщение об ошибке, ключ к источнику неприятности обычно находится на нижней строке (так как в ней то, что вы ввели) и на верхней строке (поскольку там то, что вызвало сообщение об ошибке). Где-нибудь там вы обычно сможете найти причину неполадок.

Что вы должны делать дальше? Если вы сейчас введете ‘H’, то получите то же самое вспомогательное сообщение о неопределенной команде, которое видели перед этим. Если вы нажмете ⟨return⟩, TEX продолжит работу и получит результат, фактически идентичный результату эксперимента 2. Другими словами, общепринятые ответы не научили бы вас ничему новому. Поэтому напечатайте теперь ‘E’; это прервет обработку файла и подготовит для вас способ исправить ошибку. (В некоторых системах TEX вызовет стандартный текстовый редактор, и вы окажетесь на правильном месте, чтобы удалить ‘\ERROR’. В других системах TEX просто скажет вам редактировать строку 3 в файле story.tex.)

Когда вы будете еще раз редактировать `story.tex`, вы заметите, что строка 2 все еще содержит `\vship`; тот факт, что вы сказали TEX'у вставить `\vskip`, не означает, что ваш файл как-нибудь изменен. Вообще, вы должны исправить все ошибки во входном файле, которые опознаны TEX'ом во время прогона. С помощью протокольного файла удобно запомнить, что это были за ошибки.

Ну вот, эта глава в самом деле была длинной, поэтому давайте подытожим то, что было сделано. Проделав пять экспериментов, вы научились сначала (1) получать работу, напечатанную TEX'ом; (2) создавать файл, который содержит полную входную рукопись для TEX'а, (3) изменять формат начального TEX'а для получения колонок различной ширины и (4) избегать паники, когда TEX выдает суровые предупреждения.

Поэтому вы можете теперь прервать чтение этой книги и начать печатать кипы документов. Лучше, однако, продолжить взаимоотношения с автором (возможно, немного отдохнув), поскольку пока вы только лишь на пороге возможности делать много большее. И уж во всяком случае вы должны прочитать главу 7, потому что она предупреждает о неких символах, которые вы не должны печатать, разве что в специальных целях. Когда будете читать оставшиеся главы, конечно, лучше продолжить экспериментальные запуски, используя экспериментальные тексты собственной конструкции.

Тому, что мы должны научиться делать,
мы учимся, делая.

What we have to learn to do
we learn by doing.

— ARISTOTLE, *Ethica Nicomachea* II (c. 325 B.C.)

Ему дано читать быстротекущее.

He that runs may read.

— WILLIAM COWPER, *Tirocinium* (1785)

7

Как TEX читает то, что вы вводите

В предыдущей главе мы видели, что входная рукопись выражается в терминах "строк", но что эти входные строки, по-существу, не зависимы от выходных строк, которые будут появляться в результате на страницах. Таким образом, когда вы подготавливаете или редактируете файл, можно прервать входную строку в том месте, которое вам удобно. Также следует упомянуть и несколько других относящихся к делу правил:

- ⟨return⟩ — это то же самое, что пробел.
- Два пробела рядом считаются за один пробел.
- Пустая строка означает конец абзаца.

Строго говоря, эти правила противоречивы: если ввести ⟨return⟩ дважды подряд, то получается пустая строка, а это отличается от введенных подряд двух пробелов. Когда-нибудь вас заинтересуют *точные* правила. В этой и следующей главах мы будем изучать только самую первую стадию перехода от входа к выходу.

Во-первых, необходимо получить точное представление о том, что ваша клавиатура посылает машине. Существуют 128 символов, с которыми TEX сталкивается на каждом шагу, в файле или в строке текста, введенной прямо с терминала. Эти 128 символов подразделены на 16 категорий с номерами от 0 до 15:

Номер	*Значение*	
0	Сигнальный символ	(в этом руководстве \)
1	Начало группы	(в этом руководстве {)
2	Конец группы	(в этом руководстве })
3	Математический ключ	(в этом руководстве $)
4	Табулятор	(в этом руководстве &)
5	Конец строки	(в этом руководстве ⟨return⟩)
6	Параметр	(в этом руководстве #)
7	Верхний индекс	(в этом руководстве ^)
8	Нижний индекс	(в этом руководстве _)
9	Игнорируемый символ	(в этом руководстве ⟨null⟩)
10	Пробел	(в этом руководстве ␣)
11	Буква	(A — Z, a — z, А — Я, а — я)
12	Другой символ	(не перечисленное выше или ниже)
13	Активный символ	(в этом руководстве ~)
14	Символ комментария	(в этом руководстве %)
15	Ошибочный символ	(в этом руководстве ⟨delete⟩)

Нет необходимости заучивать эти номера; важно только, что TEX различает 16 различных типов символов. Сначала это руководство заставило вас поверить, что есть только два типа — сигнальный символ и остальные, затем вам сказали еще о двух типах, символах группирования { и }. В главе 6 вы изучили еще два: ~ и %. Теперь вы знаете, что в действительности их 16.

Это уже полная правда, неописанных типов больше не осталось. Номер категории любого символа можно изменить в любое время, но обычно разумно придерживаться определенной схемы.

Главное, что следует иметь ввиду — то, что каждый формат TEX'а резервирует некоторые символы для собственных специальных целей. Например, когда вы используете формат начального TEX'а (приложение B), вам надо знать, что десять символов

```
\ { } $ & # ^ _ % ~
```

не могут быть использованы обычным образом, так как каждый из них указывает на какие-нибудь специальные действия TEX'а, как это объясняется где-нибудь в этой книге. Если вам в рукописи нужны эти символы, начальный TEX дает возможность ввести

`\$` для $, `\%` для %, `\&` для &, `\#` для #, `\_` для _;

символ `\_` полезен для *составных_идентификаторов* в компьютерных программах. В математических формулах можно использовать `\{` и `\}` для обозначения { и }, в то время как команда `\backslash` производит обратную косую черту; например,

`$\{a \backslash b\}$` дает $\{a\backslash b\}$.

Далее, `\^` ставит над буквой шляпку (циркумфлекс) (например, `\^e` дает ê), а `\~` — волну (тильду) (например, `\~n` дает ñ).

▸ **Упражнение 7.1**
Какие ужасные ошибки появятся в следующем предложении?

```
Procter & Gamble's stock climbed to $2, a 10% gain.
```

▸ **Упражнение 7.2**
Можете вы догадаться, почему разработчик начального TEX'а решил не делать `\\` командой для обратной косой черты?

Когда TEX читает строку текста из файла или строку текста, которую вы вводите непосредственно с терминала, он превращает этот текст в список "элементов". Элемент — это либо (a) простой символ с номером его категории, либо (b) команда. Например, если действуют соглашения начального TEX'а, текст '`{\hskip 36 pt}`' преобразуется в список из восьми элементов:

$\texttt{\{}_1$ $\boxed{\texttt{hskip}}$ $\texttt{3}_{12}$ $\texttt{6}_{12}$ $\texttt{␣}_{10}$ $\texttt{p}_{11}$ $\texttt{t}_{11}$ $\texttt{\}}_2$

Нижние индексы здесь — это номера категорий, как они перечислялись ранее: 1 для "начала группы", 12 для "другого символа" и так далее; $\boxed{\texttt{hskip}}$ не имеет нижнего индекса, потому что представляет собой элемент-команду, а не элемент-символ. Заметим, что пробел после `\hskip` не занесен в список элементов, потому что он следует за командным словом.

Если вы хотите достигнуть полного понимания TEX'а, важно понять идею списка элементов. Это понятие удобно изучать, если представить TEX, как живой организм. Индивидуальные строки входа в вашем файле различают только "глаза" и "пасть" TEX'а; но после того, как текст "проглочен", он посылается в "желудок" TEX'а в виде списка элементов. Пищеварительные процессы, которые собственно и производят набор, основаны целиком на элементах. Что касается желудка, входные данные впадают в него как поток элементов, как если бы ваша рукопись была напечатана вся в одну сверхдлинную строку.

О элементах TEX'а вы должны помнить две основные вещи. (1) Команда рассматривается как один объект, который больше не состоит из последовательности символов. Поэтому для TEX'а не труднее иметь дело с длинными именами команд, чем с короткими, после того, как они заменены элементами. Более того, пробелы после команд внутри списка не игнорируются; правило игнорирования пробелов применяется только во входном файле в то время, когда строка символов превращается в элементы. (2) Как только номер категории прикреплен к символьному элементу, это прикрепление остается постоянным. Например, если символ `{` вдруг объявлен как символ категории 12 вместо категории 1, символ $\texttt{\{}_1$ во внутреннем списке элементов TEX'а будет все еще оставаться с категорией 1; только заново составленный список будет содержать элементы $\texttt{\{}_{12}$. Другими словами, конкретные символы получают фиксированную интерпретацию, как только они прочитаны из файла, и эта интерпретация основана на категории, которая действовала во время чтения. Другое дело команды, они могут изменять свою интерпретацию в любое время. Пищеварительные процессы TEX'а всегда точно знают, что означает символьный элемент, поскольку номер категории появляется в самом элементе, но когда пищеварительные процессы сталкиваются с командными элементами, они должны найти текущее определение этой команды, чтобы вычислить, что она означает.

Упражнение 7.3

Некоторые номера категорий от 0 до 15 никогда не появляются в качестве индекса в символьном элементе, потому что исчезают в пасти TEX'а. Например, символы категории 0 (сигнальные символы) никогда не станут элементами. Какие категории могут реально достичь желудка TEX'а?

Существует программа, называемая `INITEX`, которая используется для начальной установки TEX'а; `INITEX` похожа на TEX, за исключением того, что может делать еще кое-что. Она может собрать образцы переноса в специальные таблицы, что ускоряет подбор переносов, а также создавать форматные файлы типа `plain.fmt` из `plain.tex`. Но `INITEX` для выполнения таких задач нуждается в дополнительной памяти, поэтому для набора обычно остается меньше памяти, чем вы рассчитывали найти в рабочей версии TEX'а.

Когда программа `INITEX` начинает работу, она не знает ничего, кроме примитивов TEX'а. Всем 128 символам присвоена категория 12, за исключением того, что ⟨return⟩ имеет категорию 5, ⟨space⟩ — категорию 10, ⟨null⟩ — категорию 9, ⟨delete⟩ —категорию 15, буквы от `A` до `Z`, от `a` до `z`, от `А` до `Я` и от `а` до `я` — категорию 11, а `%` и `\` — соответственно, категории 14 и 0. Отсюда следует, что программа `INITEX` первоначально не способна выполнить некоторые примитивы TEX'а, которые зависят от группирования; она не может использовать

`\def` или `\hbox`, когда в них есть символы категорий 1 и 2. Для того, чтобы определить символы необходимых категорий, формат в приложении B начинается с команд `\catcode`. Например,

```
\catcode`\{=1
```

присваивает категорию 1 символу `{`. Операция `\catcode` похожа на многие другие примитивы TEX'а, которые мы будем изучать позднее. Изменяя внутренние величины типа номера категорий, вы можете адаптировать TEX к широкому разнообразию приложений.

▸ **Упражнение 7.4**
Предположим, что команды

```
\catcode`\<=1 \catcode`\>=2
```

появляются рядом с началом группы, которая начинается с `{`; эти команды дают TEX'у инструкции рассматривать `<` и `>` как ограничители группы. В соответствии с правилами TEX'а о локальности, символам `<` и `>` будут возвращены их прежние категории, когда группа окончится. Но что окончит группу, `}` или `>`?

Хотя команды и рассматриваются как простые объекты, у TEX'а есть способ преобразовать их в списки символьных элементов: если ввести `\string\cs`, где `\cs` любая команда, то получится список символов для имени этой команды. Например, `\string\TeX` образует четыре элемента: $\texttt{\textbackslash}_{12}$, $\texttt{T}_{12}$, $\texttt{e}_{12}$, $\texttt{X}_{12}$. Каждый символ в этом списке элементов автоматически получает номер категории 12 ("другие"), включая и обратную косую черту, которую `\string` вставляет, чтобы представить сигнальный символ. Однако, символу '␣' (пробел) будет присвоена категория 10, если этот символ как-нибудь вкрадется в имя команды.

И наоборот, вы можете перейти от списка символьных элементов к команде, сказав `\csname`⟨элементы⟩`\endcsname`. Элементы, которые появляются в этой конструкции между `\csname` и `\endcsname` могут включать в себя другие команды, при условии, что эти команды в конце концов раскроются в символы вместо примитивов TEX'а. Окончательные символы могут быть любой категории, не обязательно буквами. Например, `\csname TeX\endcsname`, по-существу, то же самое, что `\TeX`; но `\csname\TeX\endcsname` незаконно, потому что `\TeX` раскрывается в элементы, содержащие кроме букв примитив `\kern`. Вариант `\csname\string\TeX\endcsname` еще хуже, так как создаст команду `\\TeX`, т.е. элемент $\boxed{\texttt{\textbackslash TeX}}$, которую нельзя ввести обычным образом.

▸ **Упражнение 7.5**
Поэкспериментируйте с TEX'ом, чтобы посмотреть, что делает `\string`, когда за ним следует активный символ типа `~`. (Активные символы ведут себя так же, как команды, но им не предшествует сигнальный символ.) Как просто провести этот эксперимент, используя ввод с терминала, а не из файла? Какую команду можно поставить после `\string`, чтобы получить один символьный элемент $\texttt{\textbackslash}_{12}$?

▸ **Упражнение 7.6**
Какие элементы производит

```
\expandafter\string\csname a\string\ b\endcsname
```

(Здесь перед `b` три пробела. Глава 20 объясняет `\expandafter`.)

▸ **Упражнение 7.7**
Когда `\csname` используется, чтобы определить команду в первый раз, эта команда, пока она не переопределена, эквивалентна `\relax`. Используйте этот факт для того, чтобы сконструировать такую макрокоманду `\ifundefined#1`, чтобы, например,

`\ifundefined{TeX}`⟨истинный текст⟩`\else`⟨ложный текст⟩`\fi`

раскрывалась в ⟨истинный текст⟩, если либо `\TeX` не был предварительно определен, либо `\TeX` при помощи `\let` установлен равным `\relax`; в других случаях она должна раскрываться в ⟨ложный текст⟩.

В примерах, приведенных до сих пор, `\string` превращала команды в списки элементов, которые начинались с $\backslash_{12}$. Но этот элемент обратной косой черты в действительности не встроен в TEX; существует параметр, называемый `\escapechar`, который указывает, какой символ должен использоваться, когда команда выводится в виде текста. Значением `\escapechar` обычно является внутренний код TEX'а для обратной косой черты, но это может быть изменено, если желательно другое соглашение.

У TEX'а есть еще две элементопроизводящие операции, похожие на команду `\string`. Если вы напишите `\number`⟨число⟩, то получите десятичный эквивалент числа, а если напишите `\romannumeral`⟨число⟩, то получите число, записанное римскими цифрами нижнего регистра. Например, `\romannumeral24` дает `xxiv`, список из четырех элементов, каждый из которых имеет категорию 12. Операция `\number` является лишней, когда применяется к точной константе (например, `\number24` дает `24`), но она подавляет ведущие нули, а также может использоваться с числами, которые находятся во внутренних регистрах TEX'а, или с параметрами. Например, `\number-0015` дает `-15`, а если регистр `\count5` содержит значение 316, то `\number\count5` дает `316`.

Парные операции

`\uppercase{`⟨список элементов⟩`}` и `\lowercase{`⟨список элементов⟩`}`

просматривают данный список элементов и переводят все символьные элементы в их "верхний" или "нижний" эквивалент. И вот как: каждый из 128 возможных символов имеют два связанных значения, называемых `\uccode` и `\lccode`; эти значения можно изменить, так же, как и `\catcode`. Превращение в символ верхнего регистра означает, что символ заменяется значением его `\uccode`, если только значение `\uccode` не нулевое (когда изменений не производится). Превращение в символ нижнего регистра аналогично, используя `\lccode`. Номера категорий при этом не меняются. Когда `INITEX` начинает работу, все значения `\uccode` и `\lccode` равны нулю, за исключением того, что буквы от `a` до `z` под действием `\uccode` приобретают значения от `A` до `Z`, а буквы от `A` до `Z` под действием `\lccode` приобретают значения от `a` до `z`.

TEX выполняет преобразования `\uppercase` и `\lowercase` в своем желудке, а операции `\string`, `\number`, `\romannumeral` и `\csname` выполнятся по пути в желудок (как макрорасширение), как это объяснено в главе 20.

▸ **Упражнение 7.8**
Какой список элементов получается из

`\uppercase{a\lowercase{bC}}` ?

▸ **Упражнение 7.9**
TEX имеет внутренний целочисленный параметр, называемый `\year`, который устанавливается в начале каждой работы равным текущему году. Объясните, как использовать `\year` вместе с `\romannumeral` и `\uppercase` для того, чтобы печатать для всех работ в 2003 году такой символ авторского права:

© MCMLXXXVI.

▸ **Упражнение 7.10**
Определите такую команду `\appendroman` с тремя параметрами, что `\appendroman#1#2#3` преобразовывает команду `#1` в команду, имя которой — это имя команды `#2`, за которым следует целое положительное `#3`, выраженное римскими цифрами. Например, предположим, что регистр `\count20` равен 30; тогда

```
\appendroman\a\TeX{\count20}
```

должно действовать также, как `\def\a{\TeXxxx}`.

Некоторые книги следует пробовать на вкус,
другие — проглатывать,
и немногие — жевать и переваривать.

Some bookes are to bee tasted,
others to bee swallowed,
and some few to bee chewedand disgested.
— FRANCIS BACON, *Essayes* (1597)

Именно наличие хорошего читателя
определяет хорошую книгу.

'Tis the good reader
that makes the good book.
— RALPH WALDO EMERSON, *Society & Solitude*(1870)

8

Вводимые символы

TEX применяется на машинах с различными клавиатурами, но не все из этих клавиатур содержат 128 различных символов. К тому же, как мы видим, некоторые символы, которые вы *можете* нажать на вашей клавиатуре, зарезервированы для специальных целей, таких, как, например, сигналы операционной системе и группирование. Однако, когда мы изучали шрифты, было отмечено, что в каждом шрифте имеется 256 символов. Итак, как же вы можете ссылаться на символы, которых нет на вашей клавиатуре или на те, которые были заранее задействованы для форматирования?

Один ответ — это использовать команды. Например, начальный формат приложения B, который определяет `%` как символ конца строки, так что вы можете использовать его для комментариев, определяет, что команда `\%` означает знак процента.

Для того, чтобы получить доступ к какому-нибудь символу, вы можете ввести

`\char`⟨число⟩

где ⟨число⟩ — это любое число от 0 до 255 (с необязательным пробелом после него); тогда вы получите соответствующий символ текущего шрифта. В приложении B с `\%` поступают так. Там дается определение, что `\%` — это сокращение для `\char37`, поскольку 37 — код символа "знак процента".

Кодировка, которую TEX использует для внутреннего представления символов, основана на "ASCII" (American Standard Code for Information Interchange). Приложение C дает полное описание этой кодировки, в которой некоторым управляющим функциям присвоены числа так же, как обычным буквам и знакам пунктуации. Например, ⟨space⟩ = 32, а ⟨return⟩ = 13. Существует 94 стандартных видимых символов, и им присвоены числовые коды от 33 до 126 включительно.

Оказывается, что символ `b` в ASCII имеет числовой код 98. Так что, если клавиша 'b' на вашей клавиатуре сломалась, вы можете ввести слово `bubble` в рукописи таким странным способом:

```
\char98 u\char98\char98 le
```

(Необязательный пробел после константы типа `98` игнорируется. Конечно, чтобы напечатать `\char`, вам потребуются клавиши `\`, `c`, `h`, `a`, и `r`, поэтому давайте будем надеяться, что они всегда исправны.)

Для внутреннего представления символов TEX всегда использует кодировку приложения C, независимо от того, какая кодировка применялась во входном текстовом файле. Так, `b` внутри TEX'а всегда 98, даже когда ваш компьютер обычно работает с EBCDIC или с какой-нибудь другой кодировкой, отличной от ASCII; математическое обеспечение TEX'а преобразовывает текстовые файлы во внутреннюю кодировку и назад во внешнюю кодировку, когда создает выходные текстовые файлы. Независимые от устройства (`dvi`) выходные файлы используют внутреннюю кодировку TEX'а. Поэтому TEX способен на разных компьютерах давать одинаковые результаты.

Таблицы символьных кодов типа приведенных в приложении C часто содержат числовые коды в *восьмеричной записи*, т.е., в восьмеричной системе исчисления, цифрами которой являются *0*, *1*, *2*, *3*, *4*, *5*, *6*, *7*.* Иногда также используется *шестнадцатеричная запись*, в которой в качестве цифр используются `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `A`, `B`, `C`, `D`, `E`, `F`. Например, восьмеричный код для `b` равен *142*, а его шестнадцатиричный код — `62`. ⟨Число⟩ на языке TEX'а может начинаться с `'`, и тогда оно рассматривается как восьмеричное, или с `"` — тогда оно считается шестнадцатеричным. Таким образом, `\char'142` и `\char"62` эквивалентны `\char98`. Разрешенные символьные коды в восьмеричной записи расположены от *'0* до *'377*; а в шестнадцатиричной — от `"0` до `"FF`.

Но TEX предусматривает и ⟨число⟩ другого типа, которое избавляет вас от необходимости вообще знать ASCII! Элемент `‘`$_{12}$ (левая кавычка), когда за ним следует любой символ или любая команда, имя которой — одиночный символ, означает внутренний код TEX'а для символа, который вас интересует. Например, `\char‘b` и `\char‘\b` также эквивалентны `\char98`. Если вы заглянете в приложение B, чтобы посмотреть, как определен `\%`, вы заметите, что это определение таково:

```
\def\%{\char‘\%}
```

а не `\char37`, как провозглашалось выше.

▸ **Упражнение 8.1**
Что было бы неправильным в определении `\def\%{\char‘%}`?

Предисловие к этому руководству обращает внимание, что автор время от времени допускает маленькую невинную ложь. Ну вот, если вы действительно сверились с приложением B, то обнаружили, что на самом деле там такое определение `\%`:

```
\chardef\%=‘\%
```

Поскольку разработчики форматов часто хотят связать специальный символ с именем специальной команды, TEX предусматривает конструкцию

`\chardef`⟨команда⟩`=`⟨число⟩

для чисел между 0 и 255, как эффективную альтернативу для

`\def`⟨команда⟩`{\char`⟨число⟩`}`.

Хотя вы можете использовать команду `\char`, чтобы получить доступ к любому символу текущего шрифта, вы не можете использовать ее в середине команды. Например, если вы вводите

```
\\char98
```

TEX читает это как команду `\\` со следующими за ней символами `c`, `h`, `a`, и т.д., а не как команду `\b`.

---

* Автору этого руководства нравится использовать для восьмеричных цифр курсив, чтобы типографским способом выделить, когда это возможно, основание системы исчисления.

Во время ввода рукописи почти никогда не понадобится использовать `\char`, поскольку нужные вам символы, вероятно, будут доступны как предварительно определенные команды; `\char` в основном предназначен для разработчиков форматов книг типа тех, которые приведены в приложениях. Но иногда вам может потребоваться специальный символ, и тогда, может быть, в поисках его придется перерыть весь каталог шрифтов. Как только вы его найдете, то можете его использовать, просто выбирая подходящий шрифт и затем задавая номер символа при помощи `\char`. Например, знак "опасного поворота", используемый в этом руководстве, появляется как символьное число 127 шрифта `manfnt`, а этот шрифт выбирается командой `\manual`. Поэтому макрокоманды из приложения E печатают знаки опасного поворота командами `{\manual\char127}`.

Мы видели, что набор символов ASCII включает в себя только 94 печатных символов, а TEX внутри себя работает со 128 различными символьными кодами, от 0 до 127, каждому из которых присвоена одна из шестнадцати категорий, описанных в главе 7. Если ваша клавиатура имеет дополнительные символы или если в ней нет стандартных 94 символов, специалисты, которые устанавливают вашу систему TEX, могут рассказать о соответствии между тем, что вы вводите, и символьными числами, которые воспринимает TEX. Некоторым посчастливилось иметь клавиши, помеченные $\neq$, $\leq$ и $\geq$; можно установить TEX так, что он будет распознавать эти удобные символы, что сделает ввод математических текстов более приятным. Но если у вас нет таких клавиш, вы можете обойтись командами `\ne`, `\le` и `\ge`.

У TEX'а есть стандартный способ ссылаться на невидимые символы ASCII. Символ с кодом 0 может быть введен как команда из трех символов `^^@`, код 1 — как `^^A`, и так далее до кода 31, который будет вводиться как `^^_` (см. приложение C). Если символ, следующий за `^^`, имеет внутренний код больше или равный 64, TEX вычитает число 64, в противном случае TEX добавляет 64. Следовательно, код 127 может быть напечатан как `^^?`, а знака опасного поворота можно добиться, сказав `{\manual^^?}`. Однако, вы должны изменить номер категории символа 127 перед тем, как использовать его, поскольку этот символ обычно имеет категорию 15 (ошибочный); например, можно сказать `\catcode‘\^^?=12`. Запись при помощи `^^` отличается от `\char`, потому что комбинация `^^` воспринимается как один символ; например, было бы недопустимым сказать `\catcode‘\char127`, но символы `^^` можно использовать даже как букву внутри команд.

Одно из сообщений о переполнении в главе 6 иллюстрирует тот факт, что TEX иногда использует в своих результатах забавное сочетание `^^`: символ умляут в этом примере появляется, как `^^?`, седиль — как `^^X`, поскольку ¨ и ¸ занимают позиции *'177* и *'30* шрифта `\tenrm`.

Большинство кодов `^^` не существенны, кроме как в необычных приложениях. Но особого упоминания заслуживает `^^M`, потому что это — код 13, ⟨return⟩ ASCII, который TEX обычно помещает справа от каждой строки входного

файла. Изменяя категорию для `^^M`, вы можете получить полезный специальный эффект, как мы и увидим позже.

Код `^^I` также представляет потенциальный интерес, поскольку это ⟨tab⟩ в ASCII. Начальный TEX определяет ⟨tab⟩ действующим как пробел.

Тем, кто устанавливает системы TEX для использования с неамериканским алфавитами, рекомендуется для всех дополнительных букв использовать символьные коды меньшие, чем 32 и присваивать этим кодам категорию 11 (буква). Например, предположим, у вас норвежская клавиатура , которая содержит букву `æ`. Можно разработать такой интерфейс для TEX'a, что эта буква включается, скажем, как код 26,* и ваш стандартный форматный пакет должен определять `\catcode‘æ=11`. Тогда могут быть такие команды, как `\særtrykk`, и ваши входные TEXовские файлы будут читаться американскими версиями TEX'a, не имеющими такой клавиатуры, при помощи подстановки `^^Z` для символа 26. (Например, указанная команда могла бы появиться в файле в виде `\s^^Zrtrykk`; вашим американским друзьям должен быть предоставлен тот формат, который вы используете, вместе с `\catcode‘^^Z=11`.) Конечно, следует подготовить шрифт так, чтобы символ TEX'a 26 печатался как æ, и так изменить алгоритм переноса слов, чтобы он делал переносы, правильные для норвежского языка. Главное, что такие изменения не слишком трудны; ничто в конструкции TEX'a не ограничивает его американским алфавитом до тех пор, пока у вас не более 128 различных символов.

Но подождите, говорите вы. Почему символы нумеруются от 0 до 127, когда шрифты могут содержать до 256 различных символов? Ответ заключается в том, что TEX несколькими достаточно удобными способами может достичь позиций от 128 до 255, даже хотя его символы имеют коды от 0 до 127. Вы можете использовать `\char`, как уже упоминалось, обычно через дополнительную команду,как уже упоминалось, а более высокие позиции шрифта, как мы увидим позже, удобно занять под математические символы. Другой важный способ генерировать коды, превышающие 127 — делать это последовательными нажатиями клавиш (т.е., лигатурами), если шрифт правильно разработан. Часто легче быстренько напечатать последовательность букв, чем искать одну клавишу на большой клавиатуре; таким образом, ограничение в 128 печатных символов в действительности не является неразумным.

Например, давайте опять рассмотрим норвежский алфавит, но предположим, что вы хотите использовать клавиатуру без символа `æ`. Можно так переработать метрический файл шрифта, что TEX будет интерпретировать `ae`, `o/`, `aa`, `AE`, `O/` и `AA` как лигатуры, которые дают, соответственно, æ, ø, å, Æ, Ø, и Å, а также можно было бы поместить символы å и Å в позиции шрифта 128 и 129.

---

* В числе 26 нет ничего особенного, за исключением того, что Computer Modern шрифты начального TEX'a уже имеют æв позиции 26. Однако некоторое изменение макета шрифта неизбежно, поскольку все шесть специальных букв æ, ø, å, Æ, Øи Å должны быть приписаны позициям, меньшим 32. Символы, уже находящиеся в этих позициях, можно легко передвинуть на позиции, большие чем 127, поскольку они никогда не будут доступны начальному TEX'у, кроме как через команды.

Установив `\catcode‘/=11`, вы могли бы использовать лигатуру `o/` в такой команде, как `\ho/yre`. Метод TEX'а по переносу слов лигатуры не запутают, так что вы можете использовать эту схему, по существу, так же, как предлагалось ранее, но с двумя нажатиями клавиш время от времени заменяющими один. (Ваш наборщик должен следить за теми иногда встречающимися случаями, когда соседствующие символы `aa`, `oe`, и `o/` не должны трактоваться, как лигатуры; к тому же, `\/` будет командным словом , а не командным символом.)

Остаток этой главы посвящен правилам чтения TEX'а, которые определяют преобразование из текста в элементы. Например, тот факт, что TEX игнорирует пробелы после команд, является следствием приводимых ниже правил, которые, между прочим, включают в себя и то, что пробелы после команд никогда не превращаются в шифры пробелов. Правила составлены так, чтобы работать естественным образом, поэтому, может быть, вы и не захотите их читать, но когда вы общаетесь с компьютером, приятно понимать, что машина думает о том, что она делает, и мы предоставляем вам эту возможность.

Входной информацией для TEX'а является последовательность “строк”. Всякий раз, когда TEX читает строку текста из файла, или строку текста, которую вы ввели непосредственно на терминале, читающая аппаратура находится в одном из следующих трех :

Состояние N — начало новой строки;
Состояние M — середина строки;
Состояние S — пропуск пробелов.

В начале каждой строки она находится в состоянии N; основное время — в состоянии M, а после команды или пробела — в состоянии S. Между прочим, “состояния” отличаются от “мод”, которые мы будем изучать позднее; текущее *состояние* имеет отношение к глазам TEX'а, к тому как они воспринимают символы нового текста, а текущая *мода* относится к состоянию желудочно-кишечного тракта TEX'а. Большинство из того, что делает TEX, когда он преобразует символы в элементы, не зависит от текущего состояния, кроме случаев, когда обнаружены символы пробела или конца строки (категории 10 и 5).

TEX удаляет все символы ⟨пробел⟩ (номер 32), которые встречаются на правом конце входной строки. Затем он вставляет символ ⟨return⟩ (номер 13) на правом конце строки, если только вы, когда вставляли что-то при помощи ‘`I`’ во время исправления ошибки, не внесли ничего дополнительного. Заметим, что ⟨return⟩ рассматривается, как реальный символ, который является частью строки; вы можете получить специальные эффекты, изменяя его категорию.

Если TEX в любом состоянии встречает символ начала команды (категория 0), он целиком просматривает имя команды следующим образом. (a) Если на строке больше нет символов, имя пусто (как в `\csname\endcsname`). Иначе, (b) если категория следующего символа не равна 11 (буква), имя состоит из одного символа. (c) В остальных случаях имя состоит из всех букв, начинающихся с текущей и кончающейся той, которая идет перед первой не-буквой или

концом строки. Это имя заменяется шифром команды. TEX переходит в состояние S в случае (c), а также в случае (b), когда не-буква имеет категорию 10 (пробел); в остальных случаях TEX переходит в состояние M.


Если TEX в любом состоянии встречает символ верхнего индекса (категория 7), и если за этим символом следует другой такой же символ, и к тому же эти два одинаковые символы не находятся на конце строки, то они удаляются, а к следующему символу добавляется или вычитается из него число 64. (Так, `^^A` замещается одним символом, код которого равен 1, и т.д., как это объяснялось ранее.) Это замещение осуществляется также, как когда такое трио символов встречается во время шагов (b) или (c) в процедуре просмотра имени команды, описанной выше. После того, как сделано такое замещение, TEX начинает работу снова, как будто в этом месте все время присутствовал новый символ. Если символ верхнего индекса не является началом такого трио, действует следующее правило.


Если TEX встречает символ категории 1, 2, 3, 4, 6, 8, 11, 12, 13 или символ категории 7, который не первый в только что описанном трио, он преобразует символ в элемент, присваивая ему номер категории, и переходит в состояние M. Это нормальный случай; почти все непробельные символы подчиняются этому правилу.


Если TEX встречает символ конца строки (категория 5), он отбрасывает всю информацию, которая могла остаться на текущей строке. Тогда, если TEX находился в состоянии N (новая строка), символ конца строки преобразуется в шифр команды `par` (конец абзаца); если TEX находился в состоянии M (середина строки), символ конца строки преобразуется в шифр для символа 32 (␣) категории 10 (пробел), а если TEX находился в состоянии S (пропуск пробелов), символ конца строки просто теряется.


Если TEX встречает игнорируемый символ (категория 9), он просто обходит этот символ, как будто его там нет, и остается в том же состоянии.


Если TEX встречает символ категории 10 (пробел), его действия зависят от текущего состояния. Если TEX находился в состоянии N или S, символ просто пропускается, а TEX остается в прежнем состоянии. Другое дело, если TEX находился в состоянии M; тогда символ преобразуется в элемент категории 10, символьный код которого 32, и TEX переходит в состояние S. В элементе "пробел" символьный код всегда равен 32.


Если TEX встречает символ комментария (категория 14), он отбрасывает этот символ, а также любую другую информацию, которая могла оставаться на текущей строке.


Наконец, если TEX встречает ошибочный символ (категория 15), он обходит этот символ, печатает сообщение об ошибке и остается в том же состоянии.


Если TEX'у нечего более читать на текущей строке, он переходит на следующую строку и переходит в состояние N. Однако, если указано `\endinput` для файла, для которого была команда`\input`, или если этот файл окончился, TEX возвращается туда, где он был, когда была первоначально дана команда `\input`. (Дальнейшие детали `\input` и `\endinput` обсуждаются в главе 20.)

Упражнение 8.2
Проверьте понимание правил чтения TEX'a, ответив на следующие короткие вопросы: (a) Каковы различия между категориями 5 и 14? (b) Каковы различия между категориями 3 и 4? (c) Каковы различия между категориями 11 и 12? (d) Игнорируются ли пробелы после активных символов? (e) Если строка оканчивается символом комментария, игнорируются ли пробелы в начале следующей строки? (f) Может ли игнорируемый символ появиться внутри имени команды?

Упражнение 8.3
Взгляните снова на сообщение об ошибке, приведнное на странице 40. Когда TEX сообщил, что `\vship` было неопределенной командой, он напечатал две строки контекста, показывая, что это произошло в середине чтения строки 2 файла `story`. В каком состоянии был TEX во время этого сообщения об ошибке? Какой символ он намеривался прочесть следующим?

Упражнение 8.4
Заданы номера категорий формата начального TEX'a. Какие элементы получаются из входной строки `$x^2$~ \TeX ^^C`?

Упражнение 8.5
Рассмотрите входной файл, который содержит в точности три строки; на первой строке сказано '`Hi!`', в то время как другие две совершенно пустые. Какие элементы получаются, когда TEX читает этот файл, используя категории формата начального TEX'a?

Упражнение 8.6
Предположим, что действуют номера категорий начального TEX'a, за исключением того, что символы `^^A`, `^^B`, `^^C`, `^^M` принадлежат, соответственно, категориям 0, 7, 10 и 11. Какие элементы получаются из (довольно нелепой) входной строки '`^^B^^BM^^A^^B^^C^^M^^@\M␣`'? (Помните, что за этой строкой следует ⟨return⟩, который представляет собой `^^M`; а также что `^^@` означает символ ⟨null⟩, который, когда начинается `INITEX`, имеет категорию 9.)

Специальный символ, вставляемый в конец каждой строки, не обязательно должен быть символом ⟨return⟩; TEX в действительности вставляет текущее значение целого параметра, называемого `\endlinechar`, который обычно равен 13, но может быть изменен, как любой другой параметр. Если значение параметра `\endlinechar` отрицательно или больше 127, не присоединяется никакого символа, и это приводит к такому же действию, как если бы каждая строка заканчивалась `%` (т.е., символом комментария).

Поскольку можно изменять номера категорий, TEX мог бы использовать несколько различных категорий для одного и того же символа одной строки. Например, приложения D и E содержат несколько вариантов того, как заставить TEX "дословно" обрабатывать текст, так что автор мог подготовить это руководство без особых трудностей. (Постарайтесь представить процесс печатания руководства по системе TEX: обратные косые черты и другие специальные символы надо переключать туда и обратно между их нормальной категорией и категорией 12!) Чтобы получить правильное распределение категорий, требуется внимание, но зато, разумно меняя категории, вы можете сделать так, чтобы

TEX работал многими различными способами. С другой стороны, лучше не играть с номерами категорий слишком часто, так как надо помнить, что символы никогда не изменяют своих категорий после того, как превращаются в элементы. Например, когда аргументы макрокоманды рассматриваются впервые, они помещаются в список элементов, поэтому их категории фиксируются раз и навсегда. Автор преднамеренно оставил для номеров категорий числовые, а не мнемонические обозначения, чтобы не поощрять людей к частому изменению категорий командой `\catcode`, кроме как в необычных обстоятельствах.

▸ **Упражнение 8.7**
Приложение B определяет `\lq` и `\rq` как аббревиатуры для ‘ и ’ (соответственно, простые левая и правая кавычки). Объясните, почему определения

```
\chardef\lq=96     \chardef\rq=39
```

не так уж хороши.

Ибо жизнь — не абзац
А смерть, я думаю, не скобка.

for life's not a paragraph
And death i think is no parenthesis.

— E. E. Cummings, *since feeling is first* (1926)

Этот набор символьных кодов
должен облегчать общий обмен
информацией между вычислительными
системами, системами связи и
соответствующим оборудованием.
... Был рассмотрен 8-битный набор,
но необходимость иметь более 128 кодов
в общих приложениях пока
еще не очевидна.

This coded character set is to facilitate
the general interchange of information
among information processing systems,
communication systems, and
associated equipment.
... An 8-bit set was considered
but the need for more than 128 codes
in general applications was not yet evident.

— ASA SUBCOMMITTEE X3.2, *American Standard Code for Information Interchange (1963)*

9

Романские шрифты TEX'а

Когда вы начинаете вводить рукопись для TEX'а вам надо знать, какие символы имеются в наличии. Начальный формат TEX'а в приложении B основан на Computer Modern шрифтах, которые обеспечивают символы, необходимые для печати самых разнообразных документов. Сейчас самое время обсудить, что, вводя прямой текст, можно сделать при помощи начального TEX'а. Мы уже упоминали о некоторых тонкостях — например, тире и кавычки рассматривались в главе 2, а некоторые виды акцентов (ударения) появлялись в примерах в главах 3 и 6. Цель этой главы — дать более систематическую сводку возможностей, собрав все факты воедино.

Давайте начнем с правил для обычного романского шрифта (`\rm` или `\tenrm`); начальный TEX всегда будет использовать этот шрифт, если только вы специально не указали другой. Большинство символов, которые вам нужны, легко доступны и вы можете вводить их обычным способом. Нет ничего особенного в

буквах от `A` до `Z`, от `a` до `z`, от `А` до `Я` и от `а` до `я`
цифрах от `0` до `9`
общих знаках пунктуации `: ; ! ? ( ) [ ] ‘ ’ - * / . , @`

кроме того, что TEX считает некоторые комбинации символов лигатурами:

`ff` дает ff	`ffi` дает ffi	` `` ` дает “	`!‘` дает ¡
`fi` дает fi	`ffl` дает ffl	`''` дает ”	`?‘` дает ¿
`fl` дает fl	`--` дает –	`---` дает —	

Можно также вводить `+` и `=`, чтобы получить соответствующие символы + и =, но лучше использовать такие символы только в математической моде, т.е. заключенными между двумя знаками `$`, поскольку это указывает TEX'у вставлять пробелы, подходящие для математики. Математическая мода объясняется позже, а сейчас это только удобный случай вспомнить, что формулы и текст должны быть разделены. Нематематические черточки и нематематические косые черты должны быть заданы вводом ‘`-`’ и ‘`/`’ вне математической моды, а вычитание и деление — вводом ‘`-`’ и ‘`/`’ между знаками `$`.

Предыдущий абзац охватил 80 из 94 видимых символов стандарта ASCII; но ваша клавиатура, вероятно, содержит по крайней мере еще 14 символов, и следует научиться осторожно относиться к остальным, поскольку это — специальные символы. Четыре из них зарезервированы начальным TEX'ом. Если в вашей рукописи требуются символы

`$ # % &`

вы должны помнить, что они вводятся, соответственно, как

`\$ \# \% \&`

Начальный TEX для своих целей также резервирует шесть символов

```
\   {   }   ^   _   ~
```

но вы, вероятно, не обратите внимания на их отсутствие, поскольку они в обычных рукописях не появляются. Скобки и обратные косые черты доступны через команды в математической моде.

В стандартном наборе ASCII осталось четыре специальных символа:

```
"   |   <   >
```

И снова, когда вы вводите текст, они вам реально не понадобятся. (Двойные кавычки надо заменить на ‘‘ или ’’; вертикальная черта и знаки отношения нужны только в математической моде.)

Научные публикации на английском языке часто ссылаются на другие языки, поэтому начальный TEX дает возможность вводить наиболее общеупотребительные знаки акцентов для символов из алфавитов на основе латиницы

Вводим	*получаем*	
`\‘o`	ò	(грав акцент)
`\’o`	ó	(ударение)
`\^o`	ô	(сиркомфлекс или шляпка)
`\"o`	ö	(умляут)
`\~o`	õ	(тильда)
`\=o`	ō	(макрон или надчеркивание)
`\.o`	ȯ	(точка сверху)
`\u o`	ŏ	(акцент краткости)
`\v o`	ǒ	(ха́ček или “галочка”)
`\H o`	ő	(длинный венгерский умляут)
`\t oo`	o͡o	(лига)

В пределах шрифта эти акценты разработаны так, что они появляются на высоте, правильной для буквы o; но их также можно использовать над любой другой буквой; TEX, когда надо, поднимет знак акцента. Заметим, что в последних четырех случаях необходимы пробелы, чтобы отделить команду от буквы, которая за ней следует. Однако, чтобы избежать вставки пробела внутри слова, можно напечатать `\H{o}`.

Начальный TEX также обеспечивает три вида акцента, которые расположены снизу:

Вводим	получаем	
`\c o`	o̧	(седиль)
`\d o`	ọ	(точка снизу)
`\b o`	o̲	(подчеркивание)

Есть также несколько специальных букв:

Вводим	*получаем*	
`\oe,\OE`	œ, Œ	(французская лигатура OE)
`\ae,\AE`	æ, Æ	(латинская и скандинавская лигатура AE)
`\aa,\AA`	å, Å	(скандинавская A-с-кружочком)
`\o,\O`	ø, Ø	(скандинавское O-перечеркнутое)
`\l,\L`	ł, Ł	(польская перечеркнутая L)
`\ss`	ß	(немецкая "эс-цет" или острое S)

Шрифт `\rm` содержит также бесточечные буквы ı и ȷ, которые можно получить, вводя `\i` и `\j`. Это необходимо, т.к. i и j, когда получают акцент, должны утратить свои точки. Например, чтобы правильно получить 'mīnŭs', надо напечатать '`m\=\i n\u us`' или '`m\={\i}n\u{u}s`'.

Этим завершается обзор шрифта `\rm`. Эти же самые соглашения применимы и к шрифтам `\bf`, `\sl` и `\it`, поэтому, когда вы используете другие виды печати, вам не надо делать что-либо по-другому. Например, `\bf\"o` означает **ö**, а `\it\&` означает *&*. Это приятно, не правда ли?

Однако, шрифт `\tt` немного отличается. Вам будет полезно знать, что `ff`, `fi` и тому подобное, когда вы используете шрифт пишущей машинки, не рассматриваются как лигатуры; также вы не получите лигатур из тире и знаков кавычек. Это прекрасно, потому что, когда вы пытаетесь имитировать пишущую машинку, вам нужны обычные дефисы и обычные двойные кавычки. Большинство акцентов также имеются в вашем распоряжении. Но `\H`, `\.`, `\l`, и `\L` не могут использоваться — шрифт пишущей машинки на их месте содержит другие символы. Действительно, вам неожиданно позволяется печатать `"`, `|`, `<` и `>`; см. приложение F. Все буквы, пробелы и другие символы в шрифте `\tt` имеют одну и ту же ширину.

▸ **Упражнение 9.1**
Какой не наивный способ ввести слово naïve?

▸ **Упражнение 9.2**
Назовите несколько английских слов, которые содержат акцентированные буквы.

▸ **Упражнение 9.3**
Как бы вы ввели 'Æsop's Œuvres en français'?

▸ **Упражнение 9.4**
Объясните, что надо ввести для того, чтобы получить такое предложение: *Commentarii Academiæ scientiarum imperialis petropolitanæ* стали теперь *Akademiâ Nauk SSSR, Doklady*.

▸ **Упражнение 9.5**
А как бы вы задали имена Ernesto Cesàro, Pál Erdős, Øystein Ore, Stanisław Świerczkowski, Sergeĭ Îur'ev, Muḥammad ibn Mûsâ al-Khwârizmî?

▸ **Упражнение 9.6**
Придумайте способ напечатать `Pál Erdős` шрифтом пишущей машинки.

Следующие символы выглядят одинаково независимо от того, используете ли вы шрифты `\rm`, `\sl`, `\bf`, `\it` или `\tt`:

Вводим	*получаем*	
`\dag`	†	(кинжал или обелиск)
`\ddag`	‡	(двойной кинжал или двойной крестик)
`\S`	§	(знак номера параграфа)
`\P`	¶	(знак абзаца или pilcrow)

(Они появляются в одном и том же стиле печати, потому что начальный TEX берет их из шрифта математических символов. Для математики нужно множество других символов; мы будем изучать их позже. Некоторые другие нематематические символы приведены в приложении B)

▸ **Упражнение 9.7**
В курсивном шрифте начального TEX'a знак $ получается как £. Это дает возможность ссылаться на фунт стерлингов, но вам может понадобиться и курсивный знак доллара. Можете вы придумать способ напечатать ссылку на книгу *Europe on $15.00 a day*?

Приложение B показывает, что начальный TEX обрабатывает большинство акцентов, используя примитив TEX'a `\accent`. Например, `\'#1` эквивалентно `{\accent19 #1}`, где `#1` это аргумент, который акцентируется. Общее правило таково, что `\accent`⟨число⟩ ставит акцент над следующим символом; ⟨число⟩ указывает, где этот акцент расположен в текущем шрифте. Предполагается, что акцент расположен правильно для символов, чья высота равна x-высоте текущего шрифта. Буквы вызывают поднятие или опускание акцентов с учетом наклонов акцентрирующих и акцентируемых символов. Ширина окончательной конструкции — это ширина акцентируемого символа, без учета ширины акцента. Между номером акцента и акцентируемым символом могут появиться независимые от моды команды, такие как изменение шрифта, но операции группирования туда вклиниваться не должны. Если окажется, что никакого подходящего символа нет, акцент появится сам по себе, как если бы вы сказали `\char`⟨number⟩ вместо `\accent`⟨number⟩. Например, `\'{}` дает ´.

▸ **Упражнение 9.8**
Почему, как вы думаете, начальный TEX определяет `\'#1` как `{\accent19 #1}`, вместо того, чтобы просто позволить `\'` быть сокращением для `\accent19` ? (Почему дополнительные фигурные скобки, и почему аргумент `#1`?)

Важно помнить, что те соглашения, которые мы обсуждали для акцентов и специальных букв, не встроены в сам TEX; они входят только в начальный формат TEX'a, который использует Computer Modern шрифты. Если включаются другие шрифты, будут совершенно другие соглашения; разработчики формата должны предусмотреть правила для того, чтобы в своих конкретных системах обеспечить акценты и специальные символы. Начальный TEX достаточно

хорошо работает, когда акценты не часты, но соглашения этой главы ни в коем случае не рекомендуются для широкого применения TEX'а к другим языкам. Например, хорошо разработанный шрифт TEX'а для французского языка должен, вероятно, содержать акценты в виде лигатур, так чтобы можно было печатать `e'crire de cette manie`re nai"ve en franc/ais` без обратных косых черточек. (См. примечание о норвежском языке в главе 8.)

Давайте сделаем это в торжественном романском стиле
Let's doo't after the high Roman fashion.
— WILLIAM SHAKESPEARE, *The Tragedie of Anthony and Cleopatra* (1606)

Английский язык прям, откровенен, честен, чистосердечен и не мелочен, что имеет мало общего с такими дьявольскими ухищрениями, как акценты. Действительно, редакторы и издатели в США часто приходят в ярость, когда иностранное слово прокрадывается в язык. Однако есть одно слово, за которое американцы, кажется, стоят грудью, печатая его уверенно, смело, и почти неизменно украшенным акцентом: сыр, представленный нам как Münster.

English is a straightforward, frank, honest, open-hearted, no-nonsense language, which has little truck with such devilish devious devices as accents; indeed U.S. editors and printers are often thrown into a dither when a foreign word insinuates itself into the language. However there is one word on which Americans seem to have closed ranks, printing it confidently, courageously, and almost invariably complete with accent —the cheese presented to us as Münster.

К несчастью, Munster не имеет умляута.
Unfortunately, Munster doesn't take an accent.
— WAVERLEY ROOT, in the *International Herald Tribune* (1982)

10

Размеры

Иногда надо указать TEX'у, каким сделать расстояние между строк или какой ширины должна быть строка. Например, в примере из главы 6 для пропуска по вертикали 0.5 см использовалась инструкция `\vskip .5cm`. а чтобы задать горизонтальный размер в 4 дюйма, мы говорили `\hsize=4in`. Теперь пришло время рассмотреть, какими способами можно сообщать TEX'у такие размеры.

В англоязычных странах традиционными единицами измерения являются "пункты" ("points") и "пики" ("picas"), поэтому TEX их также использует. TEX к тому же использует дюймы и обычные метрические единицы, а также версии пунктов и цицеро, принятые в континентальной Европе. Каждая единица измерения задается одним из следующих двубуквенных обозначений:

`pt`	point	пункт (расстояние между базовыми линиями в этом руководстве равно 12 pt)
`pc`	pica	пика(1 pc = 12 pt)
`in`	inch	дюйм (1 in = 72.27 pt)
`bp`	big point	большой пункт (72 bp = 1 in)
`cm`	centimeter	сантиметр (2.54 cm = 1 in)
`mm`	millimetr	миллиметр (10 mm = 1 cm)
`dd`	didot point	дидот-пункт (1157 dd = 1238 pt)
`cc`	cicero	цицеро (1 cc = 12 dd)
`sp`	scaled point	суперпункт (65536 sp = 1 pt)

Результат работы TEX'а основан на метрической системе с использованием коэффициентов преобразований, указанных здесь как точные пропорции.

▸ **Упражнение 10.1**
Сколько пунктов в 254 сантиметрах?

Когда вы хотите выразить TEX'у некий физический размер, напечатайте его как

⟨возможный знак⟩⟨число⟩⟨единица измерения⟩

или

⟨возможный знак⟩⟨строка цифр⟩.⟨строка цифр⟩
⟨единица измерения⟩

где ⟨возможный знак⟩ — это `+`, или `-` или вообще отсутствует, а ⟨строка цифр⟩ состоит из ни одной или нескольких десятичных цифр. Вместо "." также может быть и ",". Например, приведем шесть типичных размеров:

```
3 in                29 pc
-.013837in          + 42,1 dd
0.mm                123456789sp
```

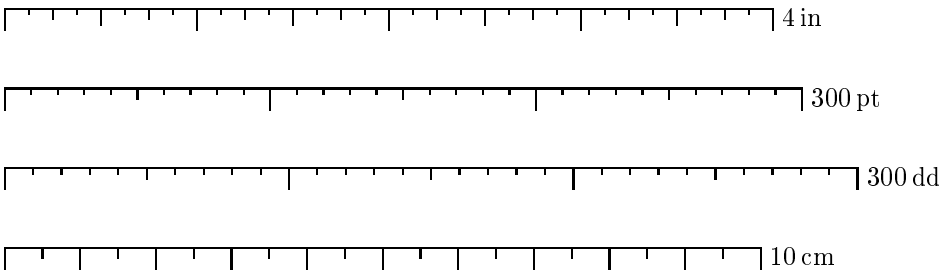
Знак плюс здесь лишний, но кое-кому время от времени нравится избыточность. Пробелы перед знаками, числами и единицами измерения не обязательны, вы можете также поставить необязательный пробел после размера, но между цифрами числа или между буквами единиц измерения пробелов быть не должно.

▸ **Упражнение 10.2**
Расположите эти шесть "типичных размеров" по порядку, от меньшего к большему.

▸ **Упражнение 10.3**
Два из следующих трех размеров разрешены правилами TEX'а. Какие? Что они означают? Почему один оставшийся неправилен?

```
'.77pt
"Ccc
-,sp
```

Приведенные ниже "линейки" напечатаны TEX'ом, так что вы можете получить некоторое представление о том, как сравнивать друг с другом различные единицы. Эти линейки очень точны, если, конечно, после завершения работы TEX'а во время копирования или печати не были внесены искажения.



▸ **Упражнение 10.4**
(Выполнять после того, как вы познакомитесь с боксами и клеем и прочтете главу 21.) Объясните, как используя TEXнапечатать такую десятисантиметровую линейку.

Внутри TEX'а все размеры представляются как целое число очень маленьких единиц,которые называются sp. Поскольку длина волны видимого света равна приблизительно 100 sp, ошибки округления в несколько sp неразличимы для глаз. Однако TEX производит все свои арифметические действия очень внимательно, так что в одном и том же документе различные реализации TEX'а приведут к одному и тому же разбиению строк и страниц, т.к. целая арифметика будет одна и та же.

Здесь единицы определены так, что точное преобразование в sp эффективно реализуется для широкого разнообразия машин. Чтобы достичь этого, "pt" сделана чуть больше, чем официальный пункт печатников, который в 1886 году был определен Американской ассоциацией учредителей печати равным точно .013837 in [сравни National Bureau of Standards Circular 570 (1956)]. Действительно, один классический пункт равен в точности .99999999 pt, поэтому "ошибка", по-существу, будет 10^{-8}. Это более чем на два порядка меньше, чем та величина, на которую за 1959 год изменился сам дюйм, когда он от своего прежнего значения (1/0.3937) cm сократился до 2.54 cm; так что о таком различии не стоит беспокоиться. Новое определение 72.27 pt = 1 in не только лучше для вычисления, но и легче для запоминания.

TEX не будет иметь дело с размерами, абсолютная величина которых больше или равна 2^{30} sp. Другими словами, максимально разрешенный размер чуть меньше, чем 16384 *pt*. Это равно около 18.892 фута (5.7583 метра), так что не ограничивает ваши возможности.

В руководствах по языку в сокращениях для различных конструкций, таких как ⟨число⟩, ⟨возможный знак⟩ и ⟨строка цифр⟩, удобно использовать "угловые скобки". Впредь для обозначения разрешенного размера TEX'а мы будем употреблять термин ⟨размер⟩. Например,

```
\hsize=⟨размер⟩
```

будет обычным способом для определения ширины колонки, которую предполагается использовать TEX'у. Смысл здесь в том, что ⟨размер⟩ может заменяться любой величиной типа `4 in`, которая удовлетворяет грамматическим правилам TEX'а для размеров; обозначение в угловых скобках облегчает изложение таких правил грамматики.

Вы должны указать единицу измерения, даже когда размер равен нулю, хотя она и кажется неуместной. Нельзя просто сказать `0`; надо сказать `0 pt`, `0 in` или что-нибудь в этом роде.

Обычно в учебниках принят размер шрифта в 10 пунктов, который вы сейчас читаете, но, возможно, вы захотите видеть и более крупный шрифт. Начальный TEX с легкостью обеспечивает увеличение результата. Если в начале рукописи сказать

```
\magnification=1200
```

все будет увеличено на 20%; т.е., получится в 1.2 раза больше нормального размера. Аналогично, `\magnification=2000` увеличивает все вдвое; это в действительности учетверяет площадь каждой буквы, поскольку удваиваются как высота, так и ширина. Для увеличения документа с коэффициентом f вы должны сказать

```
\magnification=⟨число⟩,
```

где ⟨число⟩ в 1000 раз больше f. Эта команда должна быть дана перед тем, как завершена первая страница результата. Нельзя применять к одному и тому же документу два различных увеличения.

Увеличение имеет очевидные преимущества. У вас будут меньше уставать глаза при корректировании, вы легко можете делать плакаты для лекций, а также фотографически уменьшать увеличенные результаты для того, чтобы свести к минимуму недостатки принтера с низким разрешением. И наоборот, можно сделать `\magnification=500` для того, чтобы получить карманную версию какой-нибудь книги. Но здесь маленькая ловушка: нельзя использовать увеличение, если вашему печатающему устройству не посчастливилось иметь те шрифты, которые нужны для желаемого результата. Другими словами, прежде чем увеличивать, надо убедиться, что нужные размеры имеются в наличии. Большинство реализаций TEX'а предоставляют возможность печатать все шрифты начального TEX'а, если вы увеличиваете их с помощью `\magstep0`, `1`, `2`, `3` и, возможно, `4` или даже `5` (см. главу 4); но использование больших шрифтов может оказаться дорогим, потому что для создания символов часто требуется большой объем памяти.

▸ **Упражнение 10.5**
Постарайтесь напечатать пример из главы 6 в 1.2, 1.44 и 1.728 раз больше обычного размера. Что вы должны ввести, чтобы TEX сделал это?

Когда вы говорите `\magnification=2000`, то операция `\vskip.5cm` в окончательном документе фактически будет пропускать по 1.0 cm. Если вы хотите задать размеры в терминах окончательных величин, надо непосредственно перед `pt`, `pc`, `in`, `bp`, `cm`, `mm`, `dd`, `cc` и `sp` сказать `true`. Это не увеличивает единицы, так что последующее увеличение будет отменено. Например, инструкция `\vskip.5truecm` будет эквивалентна `\vskip.25cm`, если вы перед ней сказали `\magnification=2000`. Начальный TEX сам использует эту особенность команды `\magnification`: приложение B содержит инструкцию

```
\hsize = 6.5 true in
```

сразу после того, как задействовано новое увеличение. Это устанавливает ширину строки так, что при окончательной печати материал на каждой странице будет шириной в $6\frac{1}{2}$ дюйма независимо от коэффициента увеличения. Если предположить, что бумага имеет ширину $8\frac{1}{2}$ дюйма, то как слева, так и справа будут поля шириной в 1 дюйм.

Если вы используете не "истинные" ("true") размеры, то на внутренние вычисления TEX'а не влияет присутствие или отсутствие увеличения; разбиение на строки и на страницы будет одним и тем же, а `dvi`–файл изменится только в двух местах. TEX просто говорит программе вывода, что вам нужно некоторое увеличение, а программа вывода будет делать реальное увеличение при чтении `dvi`– файла.

▸ **Упражнение 10.6**
В главе 4 упоминалось, что в одной и той же работе могут использоваться

шрифты с различными увеличениями, если их загружать в различных размерах. Объясните, какие шрифты будут использованы, когда вы даете команды

```
\magnification=\magstep1
\font\first=cmr10 scaled\magstep1
\font\second=cmr10 at 12truept
```

Увеличением в действительности управляет примитив TEX'а `\mag`, который является целым параметром и должен быть положительным и не больше 32768. Значение `\mag` проверяется в трех случаях: (1) непосредственно перед тем, как первая страница отправляется `dvi`– файл, (2) когда вычисляются `истинные` размеры; (3) когда `dvi`–файл закрывается. С другой стороны, некоторые реализации TEX'а производят не–`dvi` результаты; они проверяют `\mag` в случае (2), а также и тогда, когда выводят каждую страницу. Поскольку каждый документ имеет только одно увеличение, значение `\mag` не должно меняться после того, как оно было проверено первый раз.

TEX также распознает две единицы измерения, которые являются не абсолютными, а относительными, т.е., зависят от контекста:

`em` — это ширина "квадрата" в текущем шрифте;
`ex` — это "x-высота" текущего шрифта.

Каждый шрифт определяет свои собственные значения em и ex. В прежние времена "em" был шириной буквы M, но сейчас это уже не так; em и ex — это просто произвольные единицы, которые связаны с шрифтом. Computer Modern шрифты имеют то свойство, что двойное тире (em-dash) имеет ширину em, каждая из цифр от 0 до 9 — ширину в половину em, а строчное x — высоту, равную ex; но эти правила не являются обязательными для всех шрифтов. У шрифта `\rm` (`cmr10`) начального TEX'а 1 em = 10 pt, а 1 ex $\approx$ 4.3 pt; у шрифта `\bf` (`cmbx10`) 1 em = 11.5 pt, а 1 ex $\approx$ 4.44 pt; у шрифта `\tt` (`cmtt10`) 1 em = 10.5 pt, а 1 ex $\approx$ 4.3 pt. Все эти шрифты размером в 10 пунктов, но даже они имеют разные значения em и ex. Обычно лучше использовать `em` для горизонтальных, а `ex` — для вертикальных размеров, которые зависят от текущего шрифта.

⟨Размер⟩ может также быть внутренним регистром или параметром TEX'а. Мы будем обсуждать регистры позже и полное определение всего, что может быть размерами, будет дано в главе 24. А сейчас достаточно только упомянуть о том, что будет дальше: `\hsize` означает размер текущей горизонтальной строки, `.5\hsize` — половину этой величины, `2\wd3` обозначает двойную ширину регистра `\box3`, а `-\dimen100` — регистр `\dimen100` со знаком `-`.

Заметим, что именам единиц в размерах не предшествуют обратные косые черты. Это справедливо и для других так называемых ключевых слов языка TEX. Ключевые слова могут быть записаны заглавными буквами или вперемежку заглавными и строчными буквами, например, `Pt` эквивалентно `pt`. Номер категорий таких букв не имеет значения; вы можете, например, использовать `p` категории 12 (другие), который, как объясняется в главе 20, был генерирован раскрытием `\the\hsize`. TEX специальным образом интерпретирует ключевые слова, только если они появляются в некоторых очень ограниченных контекстах. Например, `pt` является ключевым словом, только если оно появляется после числа

в размере; `at` — ключевое слово, если оно стоит после внешнего имени шрифта в объявлении `\font`. Далее приводится полный список ключевых слов TEX'а на тот случай, если вам это любопытно:

> `at`, `bp`, `by`, `cc`, `cm`, `dd`, `depth`, `em`, `ex`, `fil`,
> `height`, `in`, `l`, `minus`, `mm`,`mu`, `pc`, `plus`, `pt`,
> `scaled`, `sp`, `spread`, `to`, `true`, `width`.

(См. приложение I для ссылок на контексты, в которых каждое из них распознается как ключевое слово.)

Методы, которые до сих пор применялись,
чтобы выяснить размер ноги римлян,
оказались на проверку настолько
неудовлетворительными, что не удивительно,
что ученые все еще не пришли
к общему мнению.

9 английских дюймов
равны 8,447 французских дюймов.

The methods that have hitherto been taken
to discover the measure of the Roman foot,
will, upon examination, be found so
unsatisfactory, that it is no wonder
the learned are not yet agreed
on that point.

...

9 London inches
are equal to 8,447 Paris inches.

— MATTHEW RAPER, in *Philosophical Transactions* (1760)

Без буквы U
units(единицы) были бы nits(пустышки).

Without the letter U,
units would be nits.

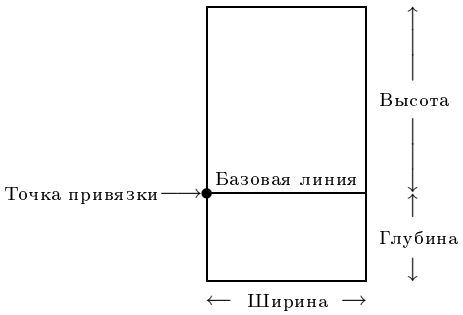
— SESAME STREET(1970)

11

Боксы

TEX составляет сложные страницы, начиная с простых символов. Он собирает их в более крупные блоки, составляя из них еще более крупные блоки, и т.д. По существу, это большая работа по склеиванию. Чтобы описать конструирование страниц, используются такие технические термины, как *боксы* и *клей*.

Боксы в TEX'е — это двумерные объекты прямоугольной формы, имеющие три объединенных измерения, называемые *высотой*, *шириной* и *глубиной*. Приведем изображение типичного бокса, которое показывает его так называемые точку привязки и базовую линию:



С точки зрения TEX'а отдельный символ шрифта является боксом; это один из простейших видов боксов. Разработчик шрифта решает, каковы высота, ширина и глубина символа, и как символ будет выглядеть, когда он в боксе. TEX использует эти размеры для того, чтобы склеить боксы вместе и в конечном счете определить местоположения точек привязки для всех символов на странице. Например, в шрифте `\rm` начального TEX'а (`cmr10`) буква h имеет высоту 6.9444 pt, ширину 5.5555 pt, а глубину равную нулю; буква g имеет высоту 4.3055 pt, ширину 5 pt, глубину 1.9444 pt. Только некоторые специальные символы, такие как круглые скобки, имеют сумму высоты и глубины действительно равную 10 pt, хотя `cmr10` и считается шрифтом в 10 пунктов. Вам не надо беспокоиться о том, чтобы самому выучить эти размеры, но было бы хорошо отдавать себе отчет в том, что TEX имеет дело с такой информацией; тогда вы сможете лучше понять, что делает компьютер с вашей рукописью.

Форма символа не обязательно должна помещаться внутри границ его бокса. Например, некоторые символы, которые используются для создания более крупных математических символов, таких, как матричные скобки, умышленно немного высовываются, так что они перекрываются с остальной частью символа. Наклонные буквы часто немного заходят за правый край бокса, как если бы бокс отклонился наверху вправо, а внизу влево, сохраняя фиксированной базовую линию. Например, сравните букву g в шрифтах

`cmr10` и `cmsl10` (`\rm` и `\sl`):

В обоих случаях TEX считает, что бокс имеет ширину в 5 pt, поэтому обе буквы обрабатываются совершенно одинаково. TEX не имеет никакого представления о том, как будут выглядеть буквы — это знает только устройство вывода. Но, несмотря на недостаток информации у TEX'а, наклонные буквы будут размещены правильно, потому что базовые линии будут выравниваться.

В действительности разработчик шрифта также сообщает TEX'у еще одну вещь, так называемую *курсивную поправку*: это число, которое задано для каждого символа и указывает, грубо говоря, насколько далеко символ выходит за правую границу своего бокса, плюс еще чуть-чуть на всякий случай. Например, курсивная поправка для g в `cmr10` равна 0.1389 pt, в то время как в `cmsl10` она 0.8565 pt. Глава 4 обращает внимание на то, что эта поправка будет добавляться к обычной ширине, если вы сразу после символа печатаете `\/`. Не следует забывать пользоваться `\/`, когда вы переключаетесь из наклонного шрифта в ненаклонный, особенно в случаях типа

```
так называемая {\sl курсивная поправка\/}:
```

поскольку для компенсации потери от наклона здесь не вставляется никакого пробела.

TEX также имеет дело с другим простым видом боксов, который можно назвать "черным боксом", а именно, с прямоугольником типа ▮, который целиком закрашивается во время печати. Для таких боксов вы также можете указать ширину, высоту и глубину, какие вам нравятся, но лучше, чтобы они не занимали много места, а то принтер может расстроиться. (Принтер вообще предпочитает белое черному).

Обычно эти черные боксы делают очень тонкими, так что они превращаются в горизонтальные или вертикальные линии. Наборщики называют такие линии "горизонтальными линейками" и "вертикальными линейками", поэтому для обозначения черных боксов TEX использует термины `\hrule` и `\vrule`. Даже когда бокс квадратный, как ■, вы должны обозвать его либо `\hrule`, либо `\vrule`. Мы будем подробно обсуждать линейчатые боксы позже. (См. главу 21.)

Все, что напечатано TEX'ом на странице, составлено из простых символьных и линейчатых боксов, склеенных в различных комбинациях. TEX склеивает боксы двумя способами, *горизонтально* и *вертикально*. Когда TEX строит горизонтальный список боксов, он располагает их так, что все точки

привязки располагаются в одном горизонтальном ряду, поэтому базовые линии соседних символов будут лежать на одной прямой. Аналогично, когда TEX создает вертикальный список боксов, он выстраивает их так, что точки привязки располагаются в одной вертикальной колонке.

Давайте посмотрим, что TEX делает за кулисами, сравнивая методы компьютера с тем, что делали бы вы, если бы собирали металлический набор вручную. По проверенному временем традиционному методу вы выбираете буквы, которые вам нужны, из коробки с шрифтами — заглавные буквы из специальной коробки — и ставите их в "наборную строку". Когда строка завершена, вы регулируете пробелы и переносите результат в "раму", где он присоединяется к другим рядам набора. В конечном итоге вы плотно запираете набор, устанавливая внешние запоры, которые называются "клиньями". Это несильно отличается от того, что делает TEX, просто используются другие слова. Когда TEX запирает строку, он создает то, что называется "ch-бокс" (горизонтальный бокс), поскольку компоненты строки собираются вместе горизонтально. Вы можете в рукописи TEX'а дать такую инструкцию:

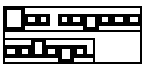
```
\hbox{Строка набора.}
```

Это указывает компьютеру взять боксы для соответствующих букв в текущем шрифте и заключить их в h-бокс. В терминах TEX'а, буква C — это бокс  , а буква p — это бокс . Поэтому данная инструкция указывает TEX'у сформировать h-бокс

который представляет из себя текст "Строка набора." Горизонтальные боксы для строк в конечном счете собирают вместе, помещая их в "v-бокс" (вертикальный бокс). Например, вы можете сказать

```
\vbox{\hbox{Две строки}\hbox{набора.}}
```

и TEX преобразует это в



т.е., Две строки
набора.

Принципиальное различие между методом TEX'а и старым способом в том, что металлический набор обычно отливается так, что каждый символ имеет одну и ту же высоту и глубину; это облегчает их выравнивание в строке вручную. В наборе TEX'а высота и глубина различны, поскольку компьютеру не трудно выравнивать символы по их базовым линиям и поскольку дополнительная информация о высоте и глубине помогает при расположении акцентов и математических символов.

Другое важное различие между набором TEX'а и ручным набором, безусловно, в том, что TEX автоматически выбирает деление строк; вы не

должны вставлять инструкции `\hbox` и `\vbox`, если только не хотите сохранить полный контроль над тем, где располагается каждая буква. С другой стороны, если вы все-таки используете `\hbox` и `\vbox`, то можете заставить TEX делать почти все, что Бен Франклин мог сделать в своей печатной мастерской. Вы только отказываетесь от возможности делать буквы изящно изогнутыми или сильно закрашенными; для таких эффектов вам надо создавать новый шрифт. (И, конечно, вы утрачиваете осязательные и обонятельные ощущения и трепетную радость от самостоятельной работы. TEX никогда полностью не заменит доброго старого способа.)

Страница текста — и та, которую вы сейчас читаете — с точки зрения TEX'а сама является боксом: это крупный бокс, составленный из вертикального списка меньших боксов, представляющих строки текста. Каждая строка текста, в свою очередь, является боксом, сделанным из горизонтального списка боксов, представляющих индивидуальные символы. В более сложных ситуациях, включающих математические формулы и/или сложные таблицы, вы можете иметь боксы внутри боксов, расположенных внутри боксов ... и так до любого уровня. Но даже эти сложные ситуации возникают из горизонтальных или вертикальных списков боксов, склеенных вместе простым способом; все, о чем вы и TEX должны беспокоиться в каждый момент времени — это один список боксов. В действительности, когда вы вводите простой текст, вы вообще не должны думать о боксах, поскольку TEX автоматически берет ответственность за сборку символьных боксов в слова, слов в строки, а строк в страницы. Вам надо осознавать понятие бокса только когда вы хотите сделать что-нибудь экстраординарное, например, когда вы хотите напечатать заголовок в центре строки.

С точки зрения пищеварительных процессов TEX'а рукопись выглядит как последовательность элементов, а элементы преобразуются в последовательность боксов. Каждый элемент входной информации является, по существу, инструкцией или частью инструкции. Например, элемент $\mathtt{A}_{11}$ обычно означает: "поставь символьный бокс для буквы `A` в конец текущего горизонтального бокса, используя текущий шрифт"; а элемент `vskip` — "пропусти вертикально в текущем вертикальном боксе величину ⟨размер⟩, указанную в следующих элементах".

Высота, ширина, или глубина бокса могли быть отрицательными, в этом случае получается "призрачный бокс", нечто, что трудно нарисовать. TEX не возражает против отрицательных размеров; он выполняет арифметические действия, как обычно. Например, составная ширина двух соседних боксов — это сумма их ширины, независимо от того, положительны они или нет. Разработчик шрифта может объявить ширину символа отрицательной, и в этом случае символ действует, как шаг назад. (Этим способом можно обращаться с языками, которые читаются справо налево, но только в ограниченной степени, поскольку алгоритм разбиения строк TEX'а основан на предположении, что слова не должны иметь отрицательную ширину).

TEX может поднимать или опускать индивидуальные боксы в горизонтальном списке; такое приспособление удобно в математических верхних

и нижних индексах, высотах акцентов, а также во многом другом. Например, далее приводится способ создания бокса, который содержит эмблему TEX, помещая его во внутренний регистр TEX'а `\box0`:

```
\setbox0=\hbox{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125em X}
```

Здесь `\kern-.1667em` означает: вставить пробел величиной в −.1667 em из текущего шрифта, т.е., чуть-чуть вернуться назад; а `\lower.5ex` означает, что бокс `\hbox{E}` надо опустить на половину текущей x-высоты, таким образом, сместить его относительно других. Вместо `\lower.5ex` можно также сказать `\raise-.5ex`. Главы 12 и 21 обсуждают подробности того, как создавать боксы для специальных эффектов; наша цель в данной главе просто получить некоторое представление об этих возможностях.

TEX может показать содержимое любого регистра боксов, если вы попросите об этом. Например, если вы введете `\showbox0` после того, как установите в `\box0` эмблему TEX, как описано выше, ваш протокольный файл будет содержать следующее мумбо-юмбо:

```
\hbox(6.83331+2.15277)x18.6108
.\tenrm T
.\kern -1.66702
.\hbox(6.83331+0.0)x6.80557, shifted 2.15277
..\tenrm E
.\kern -1.25
.\tenrm X
```

Первая строка означает, что `\box0` — это горизонтальный бокс, высота, глубина и ширина которого соответственно равны 6.83331 pt, 2.15277 pt и 18.6108 pt. Последующие строки, начинающиеся с '`.`', указывают, что они находятся *внутри* бокса. Первой в этом боксе следует буква `T` из шрифта `\tenrm`, затем идет керн. Следующим номером идет горизонтальный бокс, который содержит только букву `E`; этот бокс имеет высоту, глубину и ширину буквы `E`, и сдвинут вниз на 2.15277 pt (поэтому учитывается в глубине более крупного бокса).

▸ **Упражнение 11.1**
Почему здесь в строке '`..\tenrm E`' присутствуют две точки?

Такие демонстрации содержания боксов будут обсуждаться далее в главах 12 и 17. Они используются в основном для диагностических целей, когда вы пытаетесь точно вычислить то, что делает TEX. Основная причина того, что этот вопрос поднимается в настоящей главе — просто возможность взглянуть, как TEX представляет боксы внутри себя. Программа компьютера реально не передвигает боксы; она оперирует списками представлений боксов.

▸ **Упражнение 11.2**
Запустив TEX, вычислите, как он в действительности обрабатывает курсивную поправку к символам: как поправка представляется внутри бокса?

▸ **Упражнение 11.3**
Слово, "противоположное" TEX'у — а именно, T E X — получается при помощи

```
\setbox1=\hbox{T\kern+.1667em\raise.5ex\hbox{E}\kern+.125em X}
```

Что показала бы в этом случае команда `\showbox1`? (Постарайтесь догадаться без прогона на машине).

▸ **Упражнение 11.4**
Почему, как вы думаете, автор TEX'а не сделал боксы более симметричными относительно горизонталей и вертикалей, допустив, что точки привязки могут находиться внутри границ, а не требуя, чтобы они располагались с левого края каждого бокса?

▸ **Упражнение 11.5**
Создайте макрокоманду `\demobox` для использования в подготовке такого руководства, как это, чтобы автор мог написать `\demobox{Трудное упражнение.}` и при этом получить  .

▸ **Упражнение 11.6**
Создайте такую макрокоманду `\frac`, чтобы `\frac1/2` дало $^1/_2$.

У меня в памяти есть несколько ящиков, в которых я буду хранить их всех очень надежно, там ни один из них не будет потерян.

I have several boxes in my memory in which I will keep them all very safe, there shall not a one of them be lost.

— IZAAK WALTON, *The Compleat Angler* (1653)

Как мало любитель, сидя уютно дома, понимает заботы и тревоги автора.

How very little does the amateur, dwelling at home at ease,comprehend the labours and perils of the author.

— R. L. STEVENSON and L. OSBOURNE, *The Wrong Box* (1889)

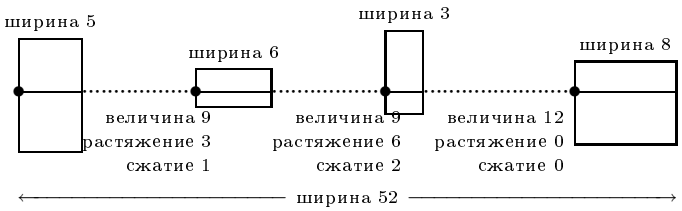
12

Клей

Но дело не только в боксах. Существует некоторый магический раствор, называемый *клеем*, который TEX использует для скрепления боксов друг с другом. Например, в этом руководстве между строками текста есть небольшое пространство; оно вычислено так, что базовые линии последовательных строк внутри абзаца отстоят друг от друга точно на 12 пунктов. Также есть пространство между словами; такое пространство — не "пустой" бокс; это — клей между боксами. Клей может растягиваться или сжиматься, так что правое поле каждой страницы получается ровным.

Когда TEX создает большой бокс из горизонтального или вертикального списка меньших боксов, между этими меньшими боксами часто располагается клей. Клей имеет три атрибута, а именно, его естественную величину (space), способность *растягиваться* (stretch) и способность *сжиматься* (shrink).

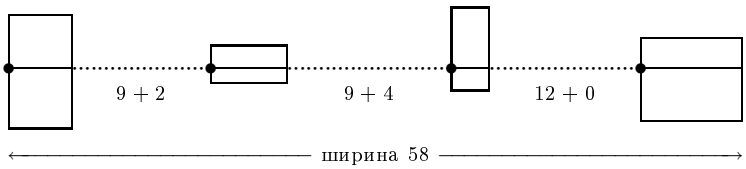
Для того, чтобы понять, как это работает, рассмотрим следующий пример горизонтального списка из четырех боксов, разделенных тремя порциями клея:



Первый элемент клея имеет величину 9 единиц, 3 единицы растяжимости и 1 единицу сжимаемости; следующий также имеет величину 9 единиц, но 6 единиц растяжимости и 2 единицы сжимаемости; последний — величину 12 единиц, но не способен ни сжиматься, ни растягиваться, поэтому остается равным 12 единицам, несмотря ни на что.

Общая ширина боксов и клея в этом примере, если рассматривать только величину клея, равна $5 + 9 + 6 + 9 + 3 + 12 + 8 = 52$ единиц. Это называется *естественной шириной* горизонтального списка и считается предпочтительным способом склеивания боксов вместе. Предположим, однако, что TEX'у сказано сделать из горизонтального списка бокс шириной 58 единиц; тогда клей нужно растянуть на 6 единиц. Далее, здесь присутствует $3+6+0 = 9$ единиц растяжимости, поэтому TEX умножает каждую единицу растяжимости на $6/9$ для того, чтобы получить дополнительно необходимые 6 единиц. Первая порция клея становится шириной $9 + (6/9) \times 3 = 11$ единиц, следующая — шириной $9 + (6/9) \times 6 = 13$ единиц, ширина последней

остается 12 единиц, и мы получаем желаемый бокс, который выглядит так:



В другом случае, если TEX'у предлагается сделать из данного списка бокс шириной в 51 единицу, клею необходимо сжаться в целом на одну единицу. Всего имеется три единицы сжимаемости, поэтому первая порция клея будет сжата на 1/3 единицы, а вторая — на 2/3 единицы.

Процесс определения величины клея во время создания бокса из горизонтальных или вертикальных списков называется *установкой клея*. Как только клей установлен, бокс становится жестким; он более не будет ни растягиваться, ни сжиматься и оказывается, по существу, неделимым.

Клей никогда не будет сжиматься больше заданной ему сжимаемости. Например, первому участку клея в нашей иллюстрации никогда не позволено быть меньше, чем 8 единиц, и TEX никогда при помощи сжатия не сделает данный горизонтальный список шириной, меньшей 49 единиц. Но если клей имеет положительную компоненту растяжимости, ему позволяется растягиваться произвольно широко.

▸ **Упражнение 12.1**
Какой ширины будут порции клея, если горизонтальный список в иллюстрации надо сделать шириной 100 единиц?

Если вы поняли концепцию клея в TEX'е, вы справедливо можете подумать, что название выбрано неудачно: настоящий клей не растягивается и не сжимается таким образом, а также не занимает заметного места между боксами, он спаивает их вместе. Слово "пружина" ближе к существу дела, поскольку пружины имеют естественную величину и поскольку различные пружины под действием напряжения сжимаются и растягиваются в различной степени. Но всякий раз, когда автор предлагал изменение терминологии TEX'а, множество людей говорили, что им нравится слово "клей", несмотря на его неподходящесть, так что первоначальное название сохранилось.

TEX неохотно растягивает клей больше заданной растяжимости, поэтому вы должны решить, насколько большой сделать каждую компоненту клея, и использовать при этом следующие правила. (a) Естественная величина клея должна быть равна промежутку, который выглядит наилучшим образом. (b) Растяжимость клея должна быть равна максимальной величине, которую можно добавить к естественному промежутку перед тем, как макет начинает выглядеть плохо. (c) Сжимаемость клея должна быть равна максимальной величине, которую можно вычесть из естественного промежутка перед тем, как макет начинает выглядеть плохо.

В большинстве случаев разработчик макета книги определяет все виды используемого клея, поэтому наборщику не надо решать, как велики должны быть атрибуты клея. Например, пользователи начального TEX'а из приложения B могут ввести `\smallskip`, когда им хочется иметь небольшой дополнительный пробел между абзацами: `\smallskip` эквивалентен участку вертикального клея в 3 pt, который может растягиваться или сжиматься на 1 pt. Вот пример `\smallskip`:

Вместо того, чтобы разбрасывать клей по всей рукописи, точно выражая величину каждого из них в пунктах, вам лучше объяснить ваши намерения более ясно, печатая что-нибудь вроде `\smallskip`, когда вы хотите получить необычный пробел. Определение `\smallskip` может быть легко изменено позже в том случае, если вы хотите такие пробелы уменьшить или увеличить. Начальный TEX также предоставляет вам `\medskip`, который равен двум smallskip'ам, и `\bigskip`, равный двум medskip'ам.

Команда `\medskip` начального TEX'а появляется в этом руководстве до и после каждой секции, помеченной знаком "опасного поворота",поэтому вы уже видели многочисленные примеры таких пробелов перед тем, как узнали, что их вызывало. Вертикальный клей создается, если вы пишите `\vskip`⟨клей⟩, где ⟨клей⟩ — это любая спецификация клея. Обычно TEX задает ⟨клей⟩ так:

⟨размер⟩ `plus`⟨размер⟩ `minus`⟨размер⟩

где `plus`⟨размер⟩ и `minus`⟨размер⟩ необязательны и в случае отсутствия предполагаются равными нулю; `plus` вводит величину растяжимости, `minus` — величину сжимаемости. Например, приложение B определяет `\medskip` как аббревиатуру для `\vskip6pt plus2pt minus2pt`. Компонента естественной величины клея должна быть всегда задана явно, даже когда ее ⟨размер⟩ равен нулю.

Горизонтальный клей создается так же, но `\vskip` заменяется на `\hskip`. Например, начальный TEX определяет команду `\enskip` как сокращение для команды `\hskip.5em\relax`; она пропускает горизонтально один "en", т.е., половину "em" текущего шрифта. У `\enskip` нет ни растяжимости, ни сжимаемости. Команда `\relax` после '`.5em`' мешает TEX'у подумать, что когда текст, следующий за `\enskip`, начинается с `plus` или `minus`, они являются ключевыми словами.

Отметим интересную вещь, которая может встретиться, когда клей растягивается и сжимается в различной степени: клей может быть *бесконечно* растяжимым. Например, рассмотрим снова те четыре бокса, которые мы видели в начале этой главы, с тем же самым клеем, что и раньше, за исключением того, что клей в середине может растягиваться бесконечно. Теперь общая растяжимость бесконечна, и когда ширина строки возрастает, все дополнительные пробелы вставляются в средний клей. Если, например, нужен бокс шириной 58, средний клей растягивается от 9 до 15 единиц, а остальные пробелы остаются неизменными.

Если такой бесконечно растяжимый клей помещен с левого края ряда боксов, это действует, как сдвиг их в крайне правое положение, т.е., максимально прижимает к правой границе создаваемого бокса. А если вы

берете *два* участка бесконечно растяжимого клея, поставив их слева и справа, это действует, как помещение списка боксов в *центр* большого бокса. Так фактически работает инструкция `\centerline` в начальном TEX'е: она помещает бесконечно растяжимый клей на обоих концах, затем создает бокс, ширина которого равна текущему значению `\hsize`.

В примере из главы 6 бесконечный клей использовался не только для центрирования, но также в инструкции `\vfill` в конце; `\vfill` по существу означает "сдвинься вертикально на нуль, но с бесконечной растяжимостью". Другими словами, `vfill` заполняет остаток текущей страницы пробелами.

TEX распознает несколько видов бесконечности, некоторые из которых "более бесконечные", чем другие. Вы можете сказать как `\vfil`, так и `\vfill`; вторая команда сильнее, чем первая. Другими словами, если не присутствует другой бесконечной растяжимости, `\vfil` растянется так, чтобы заполнить все остающееся пространство; но если присутствуют как `\vfil`, так и `\vfill` одновременно, то `\vfill` благополучно мешает `\vfil` растягиваться. Вы можете это представить, как если бы `\vfil` имел растяжимость в одну милю, а `\vfill` — в миллион миль.

Кроме `\vfil` и `\vfill`, TEX имеет `\hfil` и `\hfill` для бесконечного растяжения в горизонтальном направлении. Можно также сказать `\hss` или `\vss` для того, чтобы получить клей, который бесконечно и растягивается, и сжимается. (Имя `\hss` означает "горизонтальное растяжение и сжатие", `\vss` — его вертикальный двойник.) И наконец, примитивы `\hfilneg` и `\vfilneg` будут отменять растяжимость `\hfil` и `\vfil`. Позже мы обсудим применение этих удивительных клеев.

Приведем несколько примеров `\hfil`, используя макрокоманду начального TEX'а `\line`, которая создает h-бокс с шириной, равной текущей `\hsize`:

```
\line{Этот текст будет прижат влево.\hfil}
\line{\hfil Этот текст будет прижат вправо.}
\line{\hfil Этот текст будет в центре.\hfil}
\line{Часть текста прижата влево,\hfil а часть --- вправо.}
\line{Альфа\hfil   центр  между Альфа и Омега\hfil Омега}
\line{Пять\hfil слов\hfil на\hfil  одинаковом\hfil расстоянии.}
```

▸ **Упражнение 12.2**
Опишите результат работы следующих команд

```
\line{\hfil\hfil Что случится сейчас?\hfil}
\line{\hfill\hfil А сейчас?\hfil}
```

▸ **Упражнение 12.3**
Чем отличаются следующие три макрокоманды?

```
\def\centerlinea#1{\line{\hfil#1\hfil}}
\def\centerlineb#1{\line{\hfill#1\hfill}}
\def\centerlinec#1{\line{\hss#1\hss}}
```

Для того, чтобы задавать такие бесконечности, используя компоненты растяжимости и сжимаемости ⟨размер⟩а, можно использовать специальные единицы `fil`, `fill` и `filll`. Например, `\vfil`, `\vfill`, `\vss` и `\vfilneg`, по существу, соответственно равны спецификациям клея

```
\vskip 0pt plus 1fil
\vskip 0pt plus 1fill
\vskip 0pt plus 1fil minus 1fil
\vskip 0pt plus -1fil
```

Обычно лучше как можно чаще вставлять бесконечность первого порядка (fil), оставляя бесконечность второго порядка (fill) только для получения чего-нибудь на редкость бесконечного. Тогда бесконечность последнего порядка (filll) в случае необходимости всегда остается как последнее средство. (TEX не предусматривает примитива `\vfilll`, поскольку использование бесконечности такого наивысшего уровня не поощряется.) Вы можете использовать дробное произведение бесконечности типа `3.25fil`, если только не превышаете величину 16384 fil. TEX в действительности производит свои вычисления с целым кратным от 2^{-16} fil (или fill, или filll), поэтому `0.000007filll` оказывается неотличимым от `0pt`, но `0.00001filll` будет бесконечно больше, чем `16383.99999fill`.

Для всех TEXнических наборщиков важно знать следующее: начальный TEX увеличивает пробел в конце предложения ; более того, он автоматически увеличивает растяжимость (и уменьшает сжимаемость) после знаков пунктуации. Причина этого в том, что когда вы растягиваете строку, чтобы достигнуть заданных полей, обычно лучше поставить больший пробел после знака пунктуации, чем между двумя обычными словами. Рассмотрим, например, фразу из классического детского букваря:

```
``Oh, oh!'' cried Baby Sally. Dick and Jane laughed.
```

Если TEX устанавливает ее естественную ширину, все пробелы будут одинаковы, за исключением пробелов после кавычек и после 'Baby Sally.':

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Но если строчку необходимо расширить на 5 пунктов, 10 пунктов, 15 пунктов или более, TEX сделает так:

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Клей после запятой растягивается в 1.25 раза от величины клея между соседними словами, клей после точки и после `!''` растягивается в 3 раза. Между соседними буквами нет клея, поэтому отдельные слова будут всегда выглядеть одинаково. Если бы TEX должен был сжать эту строку до ее минимальной ширины, результат был бы такой:

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Сжимаемость клея после кавычек составляет 80 процентов сжимаемости обычного клея между словами, а после точки, восклицательного или вопросительного знака — треть обычной сжимаемости.

Все это сделано для получения красивого результата, но добавляет забот к вашей работе в качестве наборщика, потому что правило TEX'а для определения конца предложения *не всегда работает*. Дело в том, что точка иногда оказывается в середине предложения ... когда она используется (как здесь) для заполнения пропуска многоточием.

Более того, если вы попытаетесь задать '...', напечатав три точки в ряд, то получите '...' — точки слишком близки друг к другу. Один из способов справиться с этим — это перейти в *математическую* моду и использовать команду `\ldots`, определенную в формате начального TEX'а. Например, если вы вводите

```
Гммм $\ldots$ Я удивляюсь, почему?
```

то в результате будет "Гммм ... Я удивляюсь, почему?". Это получается потому, что математические формулы не подчиняются обычным правилам формирования пробелов. Глава 18 говорит подробнее об `\ldots` и связанных с этим вопросах.

Сокращения также являются проблемой. Например, рассказчик из главы 6 ссылается на Mr. Drofnats; TEX'у надо как-то указать, что точка после Mr., Mrs., Ms., Prof., Dr., Rt. Hon. и т.д. не считается знаком окончания предложения.

Мы в главе 6 обошли это затруднение, напечатав '`Mr.~Drofnats`'; знак "связки" `~` указывает начальному TEX'у вставлять обычный пробел и удерживает от разбиения строки на этом пробеле. Другой способ получить от TEX'а обычный пробел — это напечатать `\␣` (управляющий пробел); например, '`Mr.\ Drofnats`' будет почти тем же самым, что и '`Mr.~Drofnats`', за исключением того, что строчка может закончиться после 'Mr.'.

Знак связки удобно применять для сокращений внутри имени и после некоторых других общепринятых сокращений, таких как Рис., см., ср.; вы обнаружите, что легко приучить себя печатать '`см.~Рис.~5`'. Действительно, обычно разумно печатать `~` (вместо пробела) непосредственно после общепринятого сокращения, которое встречается в середине предложения. Учебники синтаксиса указываю, что за сокращением "напр." всегда должна следовать запятая, а не пробел, так что такие особые случаи не нуждаются в какой-либо специальной обработке.

Остаются только сокращения, которые очень часто возникают в библиографических ссылках; здесь подходят управляющие пробелы. Если, например, вы печатаете рукопись, которая упоминает Proc. Amer. Math. Soc, вы должны сказать

```
Proc.\ Amer.\ Math.\ Soc.
```

Согласимся, что хотя входные данные выглядят несколько некрасиво, это

приводит к правильному результату. На это время от времени приходится идти, если имеешь дело с компьютером, который пытается быть изысканным.

▸ **Упражнение 12.4**

Объясните, как напечатать следующее предложение: “Mr. & Mrs. User were married by Rev. Drofnats, who preached on Matt. 19 : 3–9.”

▸ **Упражнение 12.5**

Составьте следующую библиографическую ссылку на языке начального TEX’a: Donald E. Knuth, “Mathematical typography,” *Bull. Amer. Math. Soc.* **1** (1979), 337–372.

С другой стороны, если вы не озабочены такой утонченной расстановкой пробелов, то можете указать начальному TEX’у делать все пробелы одинаковыми, не обращая внимания на знаки пунктуации, если в начале рукописи просто напечатаете `\frenchspacing`. Это выглядит так:

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

Вы также можете переходить от одного стиля печати к другому, либо говоря `\nonfrenchspacing` для установления усложненных пробелов , либо используя `\frenchspacing` локально для некоторой группы. Например, можно использовать `\frenchspacing` только тогда, когда печатается библиография.

TEX не рассматривает точку, вопросительный или восклицательный знаки как признак конца предложения, если им предшествует заглавная буква, поскольку предполагает, что такие заглавные буквы с большой вероятностью являются чьими-нибудь инициалами. Таким образом, например, знак `\` не необходим после `I.` в `Dr.~Livingstone~I.\ Presume`; эта специфическая точка не считается концом предложения.

▸ **Упражнение 12.6**

Что вы можете сделать, чтобы заставить TeX распознавать конец предложения, которое оканчивается заглавной буквой (например, “... запущено НАСА.” или ‘Did I?’ или “... см. приложение А.”)?

Можно увидеть клей, который TEX вставляет между словами, рассматривая содержимое горизонтальных боксов во внутреннем диагностическом формате, который мы затрагивали в главе 11. Например, восклицание Бэбби Салли, после того, как TEX переварит его и вставит в бокс, предполагая режим

`\nonfrenchspacing`, начинается так:

```
.\tenrm \ (ligature ``)
.\tenrm O
.\tenrm h
.\tenrm ,
.\glue 3.33333 plus 2.08331 minus 0.88889
.\tenrm o
.\tenrm h
.\tenrm !
.\tenrm " (ligature '')
.\glue 4.44444 plus 4.99997 minus 0.37036
.\tenrm c
.\tenrm r
.\tenrm i
.\tenrm e
.\tenrm d
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm B
.\tenrm a
.\tenrm b
.\kern-0.27779
.\tenrm y
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm S
.\tenrm a
.\tenrm l
.\tenrm l
.\tenrm y
.\kern-0.83334
.\tenrm .
.\glue 4.44444 plus 4.99997 minus 0.37036
```

Нормальный клей между словами в шрифте `\tenrm` равен 3.33333 pt плюс растяжимость 1.66666 pt минус сжимаемость 1.11111 pt. Заметим, что в этом списке после знаков пунктуации клей растягивается больше, а сжимается меньше, а естественный пробел в действительности больше в конце каждого предложения. Этот пример показывает еще кое-что из того, что TEX делает при обработке этого текста. Он преобразует `` и '' в один символ, т.е. в лигатуру, и вставляет маленькие керны в два места, чтобы улучшить вид пробелов: `\kern` похож на клей, но это не тоже самое, поскольку керн не может растягиваться или сжиматься. Более того, TEX никогда не разрывает строку на керне, за исключением того, когда за керном следует клей.

Вы можете заинтересоваться, как точно действуют правила TEX'а для клея между словами. Например, как TEX запомнил действие знака восклицания Бэби Салли, когда перед следующим пробелом располагались знаки кавычек? Подробности этого несколько сложны, но не непостижимы. Когда TEX производит горизонтальный список боксов и клея, он следит за положительным

целым числом, которое называется "коэффициентом пробела".Коэффициент пробела обычно равен 1000, и это означает, что клей между словами не должен изменяться. Если коэффициент пробела f отличен от 1000, клей между словами вычисляется следующим образом. Возьмем нормальный промежуточный клей для текущего шрифта и добавим дополнительный пробел, если $f \geq 2000$. (Каждый шрифт имеет нормальный пробел, нормальную растяжимость, нормальную сжимаемость и дополнительный пробел; например, в шрифте `cmr10` эти величины равны соответственно 3.33333 pt, 1.66666 pt, 1.11111 pt и 1.11111 pt. Мы будем позднее более подробно обсуждать такие параметры шрифтов. Затем компонента растяжимости умножается на $f/1000$, а компонента сжимаемости — на $1000/f$.

Однако у TEX'а есть два параметра `\spaceskip` и `\xspaceskip`, которые позволяют отказаться от нормальных пробелов текущего шрифта. Если $f \geq 2000$ и если `\xspaceskip` не равен нулю, для пробелов между словами используется клей `\xspaceskip`. В противном случае, если `\spaceskip` не равен нулю, используется клей `\spaceskip` с компонентами растяжимости и сжимаемости, умноженными на $f/1000$ и $1000/f$. Например, макрокоманда `\raggedright` начального TEX'а для подавления растяжимости и сжимаемости пробелов между словами использует `\spaceskip` и `\xspaceskip`.

Коэффициент пробела f в начале горизонтального списка равен 1000 и опять становится равным 1000 сразу после того, как несимвольный бокс или математическая формула вставляется в горизонтальный список. Для того, чтобы назначить конкретное значение коэффициенту пробела, можно указать `\spacefactor=`⟨число⟩, но обычно f становится равным числу, не равному 1000, когда простой символьный бокс направляется в список. Каждый символ имеет код коэффициента пробела, и когда символ, код коэффициента пробела которого равен g, поступает в текущий список, коэффициенту пробела обычно присваивается новое значение, равное g. Однако, если g равно нулю, f не меняется, а если $f < 1000 < g$, коэффициент пробела устанавливается равным 1000. (Другими словами, f не перепрыгивает за один шаг от значения, меньшего 1000, к значению, большему 1000).

Когда `INITEX` создает новый формат TEX'а, все символы имеют коэффициент пробела равный 1000, за исключением заглавных букв, коэффициент пробела которых равен 999. (Это незначительное отличие и заставляет знаки пунктуации после заглавных букв действовать иначе; вы понимаете, почему?) Начальный TEX переопределяет незначительное число этих коэффициентов, используя примитив `\sfcode`, который аналогичен `\catcode` (см. приложение B). Например, инструкции

```
\sfcode‘)=0        \sfcode‘.=3000
```

делают правую скобку "прозрачной" для коэффициента пробела, в то время как растяжимость после точки утраивается. Операция `\frenchspacing` переустанавливает `\sfcode‘.` в 1000.

Когда формируются лигатуры или когда через `\char` задается специальный символ, коэффициент пробела вычисляется из отдельных символов, которые образуют лигатуру. Например, начальный TEX устанавливает коэффициент пробела для правой кавычки равным нулю, так что действие пунктуации

передается дальше. Из двух соседних символов '' формируется лигатура, которая находится в символьной позиции *'042*, но коэффициент пробела этой лигатуры двойной правой кавычки никогда не проверяется TEX'ом, поэтому начальный TEX не присваивает `\sfcode'042` какого-нибудь значения. Символы, символьный код которых больше или равен 128, должны иметь коэффициент пробела равный 1000, поскольку TEX поддерживает изменяемые `\sfcode` только для символов от 0 до 127.

Упражнение 12.7
Чему равен коэффициент пробела после каждого элемента в примере о Дике и Джейн?

Приведем способ, которым действует TEXпри распределении клея, когда упаковывает h-бокс. Естественная ширина x содержимого бокса определяется сложением ширины находящихся внутри боксов и кернов с естественной шириной всего внутреннего клея. Далее вычисляется общая сумма растяжимости и сжимаемости клея в боксе. Давайте скажем, что общая растяжимость равна $y_0 + y_1$ fil$+ y_2$ fill$+ y_3$ filll, а сжимаемость равна $z_0 + z_1$ fil$+ z_2$ fill$+ z_3$ filll. Теперь естественная ширина x сравнивается с желаемой шириной w. Если $x = w$, весь клей получает свою естественную ширину. Иначе клей будет изменяться при помощи заданной "пропорции клея" r и "порядка клея" i следующим образом. (a) Если $x < w$, TEX пытается растянуть содержимое бокса; порядок клея — это такой наибольший индекс i, что y_i не равно нулю, а пропорция клея — это $r = (w - x)/y_i$. (Если $y_0 = y_1 = y_2 = y_3 = 0$, то растяжимость отсутствует, и как i, так и r устанавливаются в нуль.) (b) Если $x > w$, TEX пробует сжать содержимое бокса аналогичным образом; порядок клея — это наибольший индекс i, при котором $z_i \neq 0$, а пропорция клея, естественно, $r = (x - w)/z_i$. Однако в случае $i = 0$ и $x - w > z_0\ r$ устанавливается равным 1.0, потому что нельзя превышать максимальную растяжимость. (c) И наконец, модифицируется каждый участок клея в горизонтальном списке бокса. Предположим, что клей имеет естественную ширину u, растяжимость y и сжимаемость z, где y — бесконечность j-того порядка, а z — бесконечность k-того порядка. Тогда, если $x < w$ (растяжение), то при $j = i$ новая ширина клея получается равной $u + ry$, а при $j \neq i$ сохраняется равной естественной ширине u. Если $x > w$ (сжатие), то при $k = i$ этот клей получает новую ширину $u - rz$, а при $k \neq i$ сохраняет естественную ширину u. Заметим, что растяжение или сжатие происходит только тогда, когда клей имеет наивысший порядок бесконечности, который не отменяется.

TEX будет создавать h-бокс заданной ширины w, если вы введете команду

`\hbox to` ⟨размер⟩`{`⟨содержимое бокса⟩`}`

где w — значение величины ⟨размер⟩. Например, макрокоманда `\line`, которая обсуждалась в этой главе ранее — это просто сокращение команды `\hbox to\hsize`. TEX также позволяет задавать точную величину растяжения или сжатия. Команда

`\hbox spread`⟨размер⟩`{`⟨содержимое бокса⟩`}`

создает бокс, ширина w которого больше на заданную величину естественной величины содержимого. Например, один из боксов, показанный в этой главе ранее,

был сгенерирован так:

```
\hbox spread 5pt{``Oh, oh!'' ... laughed.}
```

В простейшем случае, когда вы хотите получить бокс, который имеет свою естественную ширину, вам не надо писать `\hbox spread 0pt`; можно просто сказать `\hbox{`⟨содержимое бокса⟩`}`.

Базовой линией содержимого h-бокса будет общая базовая линия внутренних боксов. (Более точно, это общая базовая линия, которую они составили бы, если их не поднимать и не опускать.) Высота и глубина создаваемого h-бокса определяется максимальным расстоянием, которого достигают внутренние боксы над и под их базовыми линиями. Полученный `\hbox` никогда не будет иметь отрицательной высоты или глубины, но ширина его может быть отрицательной.

▸ **Упражнение 12.8**
Допустим, что имеется `\box1` высотой 1 pt, глубиной 1 pt, шириной 1 pt и `\box2` высотой 2 pt, глубиной 2 pt, шириной 2 pt. Третий бокс формируется так:

```
\setbox3=\hbox to3pt{\hfil\lower3pt\box1\hskip-3pt plus3fil\box2}
```

Каковы высота, глубина и ширина `\box3`? Опишите положение точек привязки боксов 1 и 2 по отношению к точке привязки бокса 3.

Для вертикальных боксов (`\vbox`) процесс установки клея аналогичен тому, что происходило с горизонтальными боксами, но прежде, чем изучать операцию `\vbox`, нам надо обсудить, как TEX вертикально нагромождает боксы, что при этом их базовые линии стараются быть друг от друга на фиксированном расстоянии. Если в горизонтальном списке боксы часто касаются друг друга, то в вертикальном списке делать это обычно считается неправильным: представьте, как ужасно выглядела бы страница, если бы ее печатные строки сдвигались друг к другу ближе всякий раз, когда они не содержат высоких букв или каких-нибудь букв, которые опускаются ниже базовой линии.

TEX решает эту проблему с помощью трех примитивов `\baselineskip`, `\lineskip` и `\lineskiplimit`. Разработчик формата, выбирая значения этих трех величин, пишет

```
\baselineskip=⟨клей⟩
\lineskip=⟨клей⟩
\lineskiplimit=⟨размер⟩
```

Интерпретация этого, по существу, такова: всякий раз, когда бокс добавляется к вертикальному списку, TEX вставляет “междустрочный клей”, предназначенный для того, чтобы сделать расстояние между базовой линией нового бокса и базовой линией предыдущего бокса в точности равным значению `\baselineskip`. Но если междустрочный клей, вычисленный в соответствии с этим правилом, укажет верхней границе нового бокса находиться ближе к нижней границе предыдущего бокса, чем величина `\lineskiplimit`, то в качестве междустрочного клея используется `\lineskip`. Другими словами, расстояние между соседними базовыми линиями будет установлено равным `\baselineskip`, если только оно не приведет к слишком большому сближению боксов; в последнем случае соседние боксы будут отделяться клеем `\lineskip`.

Правила для междустрочного клея из предыдущего абзаца выполняются без учета других видов клея, которые могли присутствовать; все вертикальные пробелы, вызванные явным заданием `\vskip` и `\kern`, действуют независимо от междустрочного клея. Так, например, команда `\smallskip` между двумя строками всегда раздвигает их базовые линии больше, чем обычно, на величину `\smallskip`. Это не влияет на решение о том, используется между этими линиями клей `\lineskip` или нет.

Например, давайте предположим, что

```
\baselineskip=12pt plus 2pt
\lineskip=3pt minus 1pt
\lineskiplimit=2pt
```

(Эти значения не особенно полезны; они выбраны просто для иллюстрации правил.) Предположим далее, что бокс, глубина которого 3 pt, был только что добавлен к текущему вертикальному списку. Мы собираемся добавить новый бокс, высота которого равна h. Если $h = 5$ pt, междустрочный клей будет равен 4 pt плюс 2 pt, поскольку, когда мы прибавим h и предыдущую глубину к величине междстрочного клея, это сделает базовые линии отстоящими друг от друга на 12 pt плюс 2 pt. Но если $h = 8$ pt, междустрочный клей будет равен 3 pt минус 1 pt, поскольку `\lineskip` будет выбрана так, чтобы, когда игнорируются растяжение и сжатие, предохранить от превышения заданный `\lineskiplimit`.

Когда вы печатаете многостраничный документ, лучше так определить `\baselineskip`, чтобы он не мог растягиваться или сжиматься, потому что это придаст страницам больше единообразия. Маленькое изменение расстояния между базовыми линиями — скажем, только в пол-пункта — может привести к существенному отличию во внешнем виде страницы, поскольку это сильно влияет на соотношение белого и черного цвета. Но если вы готовите одностраничный документ, можно придать расстоянию между базовыми линиями немного растяжимости, чтобы TEX помог вам разместить результат на странице.

▸ **Упражнение 12.9**

Как задать `\baselineskip`, `\lineskip` и `\lineskiplimit`, чтобы междустрочный клей был “непрерывной” функцией высоты следующего бокса (т.е., чтобы междустрочный клей сильно не изменялся, если высота бокса меняется только чуть-чуть)?

Изучение внутреннего представления боксов и клея TEX'а должно помочь более прочному усвоению этих понятий. Приведем выдержку из вертикального списка, который создал TEX при формировании этого абзаца:

```
\glue 6.0 plus 2.0 minus 2.0
\glue(\parskip) 0.0 plus 1.0
\glue(\baselineskip) 1.25
\hbox(7.5+1.93748)x312.0, glue set -0.1496, shifted 36.0 []
\penalty 10000
```

```
\glue(\baselineskip) 2.91254
\hbox(6.14998+1.75)x312.0, glue set 0.57817, shifted 36.0 []
\penalty 150
\glue(\baselineskip) 3.10002
\hbox(6.14998+1.93748)x348.0, glue set 45.83786fil []
\penalty 10000
\glue(\abovedisplayskip) 6.0 plus 3.0 minus 1.0
\glue(\lineskip) 1.0
\hbox(50.25+2.0)x348.0 []
```

Первый `\glue` в этом примере — это `\medskip`, который предшествует каждому абзацу со знаком опасного поворота. Затем следует клей `\parskip`, который автоматически ставится перед первой строкой нового абзаца. Затем некоторый междустрочный клей величиной 1.25 pt; он вычисляется так, чтобы после добавления высоты следующего бокса (7.5 pt) и глубины предыдущего бокса сделать итог равным 11 pt. (Предыдущий бокс не показан — это нижняя строка упражнения 12.9— но мы можем вывести, что его глубина была 2.25 pt.) Следующий `\hbox` — это первая строка этого абзаца; она сдвинута вправо на 36 pt для создания абзацного отступа. Коэффициент клея для этого бокса равен -0.1496; т.е., клей внутри бокса сжат на 14.96% своей сжимаемости. (В случае растяжения перед коэффициентом, который следует за 'glue set' отсутствовал бы `-`, поэтому мы знаем, что здесь используется сжатие.) TEX в конце каждой строки с h-боксом ставит `[]`, чтобы указать, что в боксе есть что-то еще, что здесь не показано. (Содержимое бокса было бы показано полностью, если бы выше было задано `\showboxdepth`.) Значения `\penalty` используются, чтобы предотвратить плохие разрывы между страницами, как мы увидим позднее. Третий h-бокс имеет коэффициент клея, равный 45.83786, который относится к бесконечной растяжимости первого порядка (т.е., fil) — результату команды `\hfil`, неявно вставленной перед выделенным материалом, чтобы дополнить третью строку абзаца. Наконец, четвертый h-бокс, высота которого равна 50.25 pt, указывает `\lineskip` быть междустрочным клеем. Этот бокс содержит пять выделенных строк, напечатанных шрифтом пишущей машинки; они упакованы в единый бокс и находятся в конце страницы. Если вы внимательно изучите этот пример, то многое поймете во внутренней работе TEX'а.

Исключение: никакой междустрочный клей не вставляется ни до, ни после бокса, содержащего линейку. Также можно запретить междустрочный клей, если сказать между боксами `\nointerlineskip`.

TEXовская реализация междустрочного клея включает другую примитивную величину, называемую `\prevdepth`, которая обычно содержит глубину самого свежего в текущем вертикальном списке бокса. Однако, в начале вертикального списка или сразу после бокса с линейкой значение `\prevdepth` устанавливается в предохраняющее значение -1000 pt; это служит для того, чтобы подавить следующий междустрочный клей. Пользователь может изменить значение `\prevdepth` в любой момент создания вертикального списка; таким образом, например, макрокоманда `\nointerlineskip` из приложения B раскрывается просто в `\prevdepth=-1000pt`.

Приведем точные правила вычисления междустрочного клея между боксами. Предположим, что новый бокс высотой h (не бокс-линейка) дол-

жен присоединиться к нижней границе текущего вертикального списка и пусть `\prevdepth` $= p$, `\lineskiplimit` $= l$, а `\baselineskip` $= (b$ plus y minus $z)$. Если $p \le -1000$ pt, никакой междустрочный клей не добавляется. Если это не так, то, если $b-p-h \ge l$, над новым боксом будет добавлен междустрочный клей $(b-p-h)$ plus y minus z. В остальных случаях будет добавлен клей `\lineskip`. Наконец, `\prevdepth` устанавливается равным глубине нового бокса.

▸ **Упражнение 12.10**
Mr. B. L. User хотел сложить несколько боксов вместе в вертикальный список без каких-либо промежутков между ними. Он не хотел после каждого бокса говорить `\nointerlineskip`, поэтому решил установить `\baselineskip`, `\lineskip` и `\lineskiplimit` равными `0pt`. Сработает ли это?

Вертикальным аналогом `\hbox` является `\vbox`, и TEX будет выполнять команды `\vbox to`⟨размер⟩ и `\vbox spread`⟨размер⟩ приблизительно так же, как можно было ожидать по аналогии с горизонтальным случаем. Однако в этом случае есть некоторое осложнение, потому что в вертикальном направлении боксы имеют как высоту, так и глубину, тогда как в горизонтальном направлении — только ширину. Размер в команде `\vbox` относится к окончательной высоте бокса, так что, например, `\vbox to 50pt{...}` производит бокс, высота которого 50 pt; это нас устраивает, так как все, что может растянуться или сжаться внутри вертикального бокса, находится в той части, которая влияет на высоту, в то время как глубина при установке клея не меняется.

Глубиной созданного `\vbox` лучше всего считать глубину нижнего из внутренних боксов. Таким образом, вертикальный бокс схематически строится так: берется набор боксов и подготавливается так, что точки привязки располагаются на вертикальной прямой; затем точка привязки самого нижнего бокса принимается за общую точку привязки, а клей располагается так, что окончательная высота имеет некоторое заданное значение.

Это описание v-боксов упускает некоторые детали, которые возникают, когда вы рассматриваете необычные случаи. Например, можно сдвинуть боксы в вертикальном списке вправо или влево, сказав `\moveright`⟨размер⟩⟨бокс⟩ или `\moveleft`⟨размер⟩⟨бокс⟩; это аналогично возможности поднимать (`\raise`) или опускать (`\lower`) боксы в горизонтальном списке и означает, что точки привязки внутри v-бокса необязательно всегда лежат на вертикальной прямой. Более того, необходимо бороться с боксами, которые имеют слишком большую глубину, чтобы они не залезли слишком глубоко на нижнее поле страницы; позже будет указано, что кроме боксов и клея вертикальный список может содержать некоторые другие вещи, такие как штрафы и метки.

Поэтому реальные правила определения глубины сконструированного бокса оказываются несколько TEXническими. Итак, данный вертикальный список упаковывается в `\vbox`, задача — определить его естественную глубину. (1) Если вертикальный список не содержит боксов, его глубина равна нулю. (2) Если в нем есть хотя бы один бокс, но за нижним боксом следует керн или клей, возможно, вперемежку со штрафами или другими элементами, то глубина равна нулю. (3) Если в нем есть хотя бы один бокс и если за нижним боксом не следует керн или клей, глубина равна глубине этого бокса. (4) Однако, если глубина, вычисленная по правилам (1), (2) или (3) превышает `\boxmaxdepth`, глубина

будет равна текущему значению `\boxmaxdepth`. (Начальный TEX устанавливает `\boxmaxdepth` равным наибольшему возможному размеру, поэтому правило (4) не будет применяться, если только вы не задали меньшее значение. Когда правило (4) уменьшает глубину, TEX добавляет излишнюю глубину к естественной высоте бокса, по существу, сдвигая точку привязки вниз до тех пор, пока глубина не уменьшится до установленного максимума.)

Клей в v-боксе устанавливается точно также, как в h-боксе, при помощи коэффициента и порядка клея, основанных на разности между естественной шириной x и желаемой высотой w и на величинах растяжимости и сжимаемости, которые имеются в этот момент.

Ширина вычисляемого `\vbox` равна максимальному расстоянию, на которое внутренний бокс простирается вправо от точки привязки, принимая во внимание возможный сдвиг. Эта ширина всегда неотрицательна.

▸ **Упражнение 12.11**
Предположим, что `\box1` имеет высоту 1 pt, глубину 1 pt и ширину 1 pt; `\box2` — высоту 2 pt, глубину 2 pt и ширину 2 pt; `\baselineskip`, `\lineskip` и `\lineskiplimit` все равны нулю; `\boxmaxdepth` очень велико. Третий бокс формируется следующим образом:

```
\setbox3=\vbox to3pt{\moveright3pt\box1\vskip-3pt plus3fil\box2}
```

Каковы высота, глубина и ширина `\box3`? Опишите положение точек привязки боксов 1 и 2 относительно точки привязки бокса 3.

▸ **Упражнение 12.12**
В предположениях предыдущего упражнения, но с `\baselineskip=9pt minus3fil`, опишите `\box4`, если

```
\setbox4=\vbox to4pt{\vss\box1\moveleft4pt\box2\vss}
```

▸ **Упражнение 12.13**
Решите предыдущую задачу, но с `\boxmaxdepth=-4pt`.

Мы видели, что `\vbox` соединяет набор боксов в более крупный бокс, который имеет ту же базовую линию, что и нижний из его внутренних боксов. TEX имеет другую операцию, называемую `\vtop`, которая производит такой же бокс, как `\vbox`, но с базовой линией, как у верхнего из его внутренних боксов. Например,

```
\hbox{Здесь \vtop{\hbox{две строки}\hbox{текста.}}}
```

производит

Здесь две строки
текста.

Аналогично команде `\vbox to`⟨размер⟩ можно написать `\vtop to`⟨размер⟩ и `\vtop spread`⟨размер⟩, но при этом надо понимать, что означает такая конструкция. TEX выполняет `\vtop` следующим образом. (1) Первый вертикальный бокс формируется так же, как если бы `\vtop` был `\vbox`, используя все правила для `\vbox`, приведенные выше. (2) Окончательная высота x определяется равной

нулю, если первый элемент внутри нового v-бокса не бокс, либо равной высоте этого бокса. (3) Пусть h и d — это высота и глубина v-бокса в шаге (1). TEX завершает `\vtop`, передвигая точку привязки вверх и вниз, если это необходимо, так чтобы бокс имел высоту x и глубину $h+d-x$.

Упражнение 12.14
Опишите пустые боксы, которые вы получаете из `\vbox to`⟨dimen⟩`{}` и `\vtop to`⟨dimen⟩`{}`. Каковы их высота, глубина и ширина?

Упражнение 12.15
Определите макрокоманду `\nullbox#1#2#3`, которая производит бокс, высота, глубина и ширина которого заданы тремя параметрами. Бокс не должен содержать ничего, что выдавалось бы на печать.

Операцией `\vbox` лучше получать боксы с большой высотой и маленькой глубиной, в то время как `\vtop` удобнее для маленькой высоты и большой глубины. Если вы пытаетесь составить вертикальный список из больших v-боксов, вас, тем не менее, может не удовлетворить ни `\vbox`, ни `\vtop`; также может понадобиться, чтобы бокс имел две точки привязки одновременно, одну для верха и одну для низа. Если бы использовалась такая схема с двумя точками привязки, можно было бы определить междустрочный клей, основанный на расстоянии между нижней точкой привязки одного бокса и верхней точкой привязки бокса, следующего за ним в вертикальном списке. Но, увы, TEX дает вам только одну точку привязки на бокс.

Обойти это можно, используя так называемую "подпорку".Начальный TEX определяет `\strut` как невидимый бокс с нулевой шириной, который настолько высовывается над и под базовой линией, что если строка содержит такую подпорку, вам вообще не нужен междустрочный клей. (В начальном TEX'е базовые линии находятся на расстоянии 12 pt; оказывается, что `\strut` — это вертикальная линейка высотой 8.5 pt, глубиной 3.5 pt и шириной 0 pt.) Если вы ухитритесь поставить одну подпорку на верхней строке, а другую на нижней строке большого v-бокса, то в большой сборке можно получить правильное размещение пробелов, просто поместив боксы вплотную друг к другу. Например, макрокоманда `\footnote` из приложения B устанавливает подпорки в начале и конце каждой сноски, так что когда внизу некоторой страницы оказывается несколько сносок вместе, распределение пробелов будет правильным.

Если вы поняли, что такое боксы и клей, вы подготовлены к изучению макрокоманд начального TEX'а `\rlap` и `\llap`. Это сокращения для "правого перекрытия" и "левого перекрытия". Сказать `\rlap{`⟨нечто⟩`}` — это то же самое, что напечатать ⟨нечто⟩, а затем вернуться назад, как-будто вы ничего не печатали. Более точно, `\rlap{`⟨нечто⟩`}` создает бокс нулевой ширины, при этом ⟨нечто⟩ появляется справа от этого бокса (но не занимает никакого места). Макрокоманда `\llap` аналогична, но она сначала делает возврат. Иными словами, `\llap{`⟨нечто⟩`}` создает бокс нулевой ширины с ⟨нечто⟩, продолженным от этого бокса влево. Используя шрифт пишущей машинки, вы, например, можете напечатать $\neq$, указав либо `\rlap/=`, либо `/\llap=`. Можно, используя `\llap`, вставить текст на левых полях или, используя `\rlap`, на правых полях, поскольку TEX не настаивает на том, чтобы содержимое бокса было строго ограничено границами бокса.

Интересно, что `\rlap` и `\llap` также могут быть сделаны при помощи бесконечного клея. Один из способов определения `\rlap` мог бы быть таким:

```
\def\rlap#1{{\setbox0=\hbox{#1}\copy0\kern-\wd0}}
```

но делать такие длинные вычисления нет необходимости. Реальное определение в приложении B намного элегантнее, а именно:

```
\def\rlap#1{\hbox to 0pt{#1\hss}}
```

и стоит поразмыслить, почему это работает. Предположим, например, что вы выполняете `\rlap{g}`, где буква g имеет ширину 5 pt. Поскольку `\rlap` создает h-бокс шириной 0 pt, то клей, представленный `\hss`, должен сжаться на 5 pt. Далее, этот клей имеет в качестве своей естественной ширины 0 pt, но он имеет и неопределенную сжимаемость, поэтому может легко сжаться до −5 pt, и в данном случае `\rlap` делает в точности то же, что и `\hskip-5pt`.

▸ **Упражнение 12.16**
Угадайте определение `\llap`, не заглядывая в приложения A или B.

▸ **Упражнение 12.17**
(Это продолжение упражнения 12.2, но похитрее.) Опишите результат следующей команды:

```
\line{\hfil Это загадка.\hfilneg}
```

Кое-что он преувеличивал,
но в основном он говорил правду.

There was things which he stretched,
but mainly he told the truth.
— MARK TWAIN, *Huckleberry Finn*(1884)

Каждая форма существует только
из-за пространства вокруг ее.
... Следовательно, существует правильное положение
для каждой формы в каждой ситуации.
Если нам удалось найти это положение,
мы сделали свое дело.

Every shape exist only
because of the spase around it.
... Hence there is a 'right' position
for every shape in every situation.
if we succeed in finding that position,
we have done our job.
— JAN TSCHICHOLD, *Typographische Gestaltung*(1935)

13

Моды

Так же, как люди бывают в различном настроении, TEX может находиться в различных "модах". (За тем исключением, что TEX более предсказуем, чем люди.) Существует шесть мод .

- Вертикальная мода. [Построение основного вертикального списка, из которого получаются страницы выходного документа.]
- Внутренняя вертикальная мода. [Построение вертикального списка для v-бокса.]
- Горизонтальная мода. [Построение горизонтального списка для абзаца.]
- Частная горизонтальная мода. [Построение горизонтального списка для h-бокса.]
- Математическая мода. [Построение математической формулы для помещения в горизонтальный список.]
- Выделенная математическая мода. [Построение математической формулы для помещения на отдельной строке со временным прерыванием текущего абзаца.]

В простых ситуациях вам не нужно знать, в какой моде работает TEX, поскольку компьютер сам разбирается, что и как надо делать. Но когда вы получаете сообщение об ошибке, в котором, например, сказано: "`! Вы не можете делать то-то и то-то в ограниченной горизонтальной моде`", знание мод помогает объяснить, почему TEX считает вас бестолковым.

В основном TEX находится в одной из вертикальных мод, когда готовит список боксов и клея, которые будут помещены на странице вертикально один над другим; он находится в одной из горизонтальных мод, когда готовит список боксов и клея, которые будут вытянуты горизонтально один за другим с выровненными базовыми линиями, и в одной из математических мод, когда читает формулу.

Репортаж о типичной работе TEX поясняет идею моды. В начале TEX находится в вертикальной моде, готовый создавать страницы. Если, когда TEX в вертикальной моде, вы задаете клей или бокс, то этот клей или бокс помещаются на текущей странице ниже того, что было задано до этого. Например, инструкции `\vskip` в пробном запуске, который мы обсуждали в главе 6, помещают на страницу вертикальный клей, а инструкции `\hrule` помещают горизонтальные прямые линии вверху и внизу рассказа. Команды `\centerline` также производят боксы, которые включаются в основной вертикальный список, но такие боксы требуют несколько большего труда, чем линейчатые. TEX находился в вертикальной моде, когда натолкнулся на `\centerline{\bf A SHORT STORY}`, и временно перешел в частную горизонтальную моду, когда обрабатывал слова "`A SHORT STORY`"; потом пищеварительный процесс, после установки клея в `\centerline` боксе, вернулся в вертикальную моду.

Продолжим пример главы 6. TEX переключился в горизонтальную моду, как только прочитал `O` в "`Once upon a time`". Горизонтальная мода — это мода для создания абзацев . Абзац целиком (строки с 7 по 11 файла `story`) был введен в горизонтальной моде; затем текст был поделен на выходные строки подходящей ширины. Эти строки были помещены в боксы и добавлены к странице (с соответствующим междустрочным клеем между ними), а затем TEX вернулся в вертикальную моду. Буква `M` на строке 12 снова включила горизонтальную моду.

Когда TEX находится в вертикальной или внутренней вертикальной моде, первый знак нового абзаца изменяет моду на горизонтальную для продолжения абзаца. Другими словами, то, что не имеет вертикальной ориентации, указывает моде автоматически переключиться из вертикальной в горизонтальную. Это происходит, когда вы вводите любой символ, `\char`, `\accent`, `\hskip`, `\␣` `\vrule` или математический переключатель (`$`); TEX вставляет отступ текущего абзаца (indentation) и еще раз читает горизонтальный элемент так, как будто он встречается в горизонтальной моде.

Вместо неявного переключения моды можно явно приказать TEX'у перейти в горизонтальную моду, сказав `\indent` или `\noindent`. Например, если бы строка 7 в файле `story` главы 6 начиналась с

```
\indent Once upon a time, ...
```

был бы получен тот же самый результат, потому что `\indent` дал бы TEX'у инструкцию начать абзац. А если бы эта строка начиналась с

```
\noindent Once upon a time, ...
```

то в первом абзаце рассказа не был бы сделан отступ. Команда `\noindent` просто указывает TEX'у войти в горизонтальную моду, если текущая мода является вертикальной или внутренней вертикальной; `\indent` действует аналогично, но к тому же создает пустой бокс, ширина которого равна текущему значению `\parindent`, и помещает этот пустой бокс в текущий горизонтальный список. Начальный TEX устанавливает `\parindent=20pt`. Если вы говорите `\indent\indent`, то получаете двойной отступ; если же говорите `\noindent\noindent`, то второе `\noindent` ничего не делает.

▸ **Упражнение 13.1**
Если вы скажете `\hbox{...}` в горизонтальной моде, TEX создаст указанный бокс и добавит результат к текущему абзацу. Аналогично, если вы скажете `\hbox{...}` в вертикальной моде, TEX создаст бокс и добавит его к текущей странице. Что можно сделать, если вы хотите начать абзац с `\hbox`?

Когда обрабатываются простые рукописи, TEX почти все время работает в горизонтальной моде (создание абзацев) с короткими путешествиями в вертикальную моду (между абзацами). Абзац завершается, когда вы вводите `\par` или когда в рукописи встречается пустая строка, поскольку в соответствии с правилами чтения из главы 8 пустая строка преобразуется

в `\par`. Абзац также заканчивается, когда вы вводите что-либо несовместимое с горизонтальной модой. Например, достаточно команды `\vskip 1in` на строке 16 главы 6 файла `story` для того, чтобы прервать абзац после "`...beautiful documents.`"; никакого `\par` здесь не надо, поскольку `\vskip` вводит вертикальный клей, а это не может принадлежать абзацу.

Если знак начала математической моды (`$`) появляется в горизонтальной моде, TEX погружается в математическую моду и обрабатывает формулу до тех пор, пока не встретит закрывающий `$`, затем он добавляет текст этой формулы к текущему абзацу и возвращается в горизонтальную моду. Так, в примере "I wonder why?" из главы 12 TEX временно переходит в математическую моду, когда обрабатывает `\ldots`, воспринимая многоточие как формулу.

Однако, если в абзаце появляются два последовательных знака начала математической моды (`$$`), TEX прерывает абзац, в котором он находится, добавляет прочитанную часть этого абзаца к охватывающему вертикальному списку, затем обрабатывает математическую формулу в выделенной математической моде, добавляет эту формулу к охватывающему списку и возвращается в горизонтальную моду для продолжения абзаца. (Формула, которая выделяется, должна оканчиваться `$$`.) Например, предположим вы вводите

```
число $$\pi \approx 3.1415926536$$ является важным.
```

TEX между двумя `$$` переходит в выделенную математическую моду и результат, который вы получите, утверждает, что число

$$\pi \approx 3.1415926536$$

является важным.

Когда TEX находится в вертикальной или внутренней вертикальной моде, он игнорирует пробелы и пустые строки (или команды `\par`), так что вам не надо беспокоиться, что такие вещи могут изменить моду или повлиять на создаваемый документ. Командный пробел (`\␣`) будет, однако, рассматриваться как начало абзаца; абзац начнется с пробела после отступа.

Обычно лучше окончить всю работу, поставив в конце рукописи TEX'а `\bye`, что является сокращением для `\vfill\eject\end`. Команда `\vfill` переводит TEX в вертикальную моду и вставляет достаточно пробелов для того, чтобы до конца заполнить последнюю страницу; `\eject` выводит эту последнюю страницу, а `\end`, отсылает компьютер к программе завершения TEX'а.

TEX попадает во внутреннюю вертикальную моду, когда ему надо сделать что-нибудь из вертикального списка боксов (используя `\vbox`, `\vtop`, `\vcenter`, `\valign`, `\vadjust` или `\insert`). Он попадает в частную горизонтальную моду, когда вы просите его создать что-нибудь из горизонтального списка

боксов (используя `\hbox` или `\halign`). Построение бокса обсуждается в главах 12 и 21. Позже мы увидим, что между внутренней и обычной вертикальной , а также между частной и обычной горизонтальной модами разница очень мала, но они не абсолютно одинаковы, поскольку служат различным целям.

Всякий раз, когда TEX рассматривает очередной элемент входного файла, чтобы решить, что нужно делать дальше, текущая мода может повлиять на то, что означает этот элемент. Например, `\kern` в вертикальной моде указывает на вертикальный пропуск, а в горизонтальной моде — на горизонтальный пропуск; символ математического переключения `$` указывает на переход в математическую моду из горизонтальной моды, но когда он встречается в математической моде, он указывает на выход из этой математической моды; два последовательных знака (`$$`), появляясь в горизонтальной моде, инициируют выделенную математическую моду, а в частной горизонтальной моде они просто обозначают пустую математическую формулу. TEX использует тот факт, что в некоторых модах некоторые операции неуместны, для того, чтобы помочь вам избавиться от ошибок, которые могли вкрасться в рукопись. Главы с 24 по 26 объясняют подробно, что происходит с каждым возможным элементом в каждой возможной моде.

TEX часто прерывает свою работу в одной моде, чтобы решить некоторую задачу в другой, после чего начальная мода возобновляется. Например, можно в любой моде сказать `box{`; когда TEX переварит это, он приостановит все, что в это время делал, и войдет в частную горизонтальную моду. Закрывающая `}` в конце укажет на завершение h-бокса, после чего выполнение отложенной задачи возобновится. В этом смысле TEX может одновременно находиться во многих модах, но в каждый момент на вычисления влияет только самая внутренняя мода; остальные моды вытесняются из сознания TEX'а.

Чтобы ближе познакомиться с модами TEX'а, рассмотрим курьезный тестовый файл, называемый `modes.tex`, в котором задействованы все моды сразу:

```
1 \tracingcommands=1
2 \hbox{
3 $
4 \vbox{
5 \noindent$$
6 x\showlists
7 $$}$}\bye
```

Первая строка файла `modes.tex` указывает TEX'у протоколировать каждую команду, которую он получает; TEX будет выдавать диагностические данные всякий раз, когда значение `\tracingcommands` положительно. И действительно, если вы запустите TEX с файлом `modes.tex`, вы получите файл `modes.log`, который содер-

жит следующую информацию:

```
{vertical mode: \hbox}
{restricted horizontal mode: blank space}
{math shift character $}
{math mode: blank space}
{\vbox}
{internal vertical mode: blank space}
{\noindent}
{horizontal mode: math shift character $}
{display math mode: blank space}
{the letter x}
```

Смысл этого в том, что TEX сначала увидел элемент `\hbox` в вертикальной моде; это указало ему двигаться дальше и читать следующий `{`. После этого TEX перешел в ограниченную горизонтальную моду и увидел пробел, который получился из конца второй строки файла. Затем он встретил символ математического переключения (все еще в ограниченной горизонтальной моде), который указал на переключение в математическую моду; тут встретился другой пробел. Затем `\vbox` ввел внутреннюю вертикальную моду, внутри которой `\noindent` включил горизонтальную моду; два последовательных элемента `$` привели к выделенной математической моде. (Только первый `$` был показан командой `\tracingcommands`, потому что именно он указал TEX'у искать впереди следующий `$`.)

Вслед за тем, что указано выше, в `modes.log` идет `\showlists`. Это еще одна удобная команда для диагностики, которую можно использовать, чтобы выяснить то, что TEX обычно хранит в себе; она говорит TEX'у показать списки, которые обрабатываются в текущей моде и во всех тех внешних модах, в которых была прервана работа:

```
### display math mode entered at line 5
\mathord
.\fam1 x
### internal vertical mode entered at line 4
prevdepth ignored
### math mode entered at line 3
### restricted horizontal mode entered at line 2
\glue 3.33333 plus 1.66666 minus 1.11111
spacefactor 1000
### vertical mode entered at line 0
prevdepth ignored
```

В данном случае списки представляют пять уровней активности, все они присутствуют в конце строки 6 файла `modes.tex`. Текущая мода показана первой, а именно, выделенная математическая мода, которая началась на строке 5. Текущий математический список содержит один "математический" объект, содержащий букву `x` семейства 1. (Потерпите: когда вы изучите главы, посвященные математическим формулам TEX'а, вы поймете, что это значит.) Вне выделенной математической моды идет внутренняя вертикальная мода, в которую TEX вернется, когда завершится абзац, содержащий выделенную формулу. Вертикальный

список на этом уровне пуст; "`prevdepth ignored`" означает, что `\prevdepth` имеет значение ≤ -1000 pt, поэтому следующий междустрочный клей будет опущен (ср. глава 12). Математическая мода вне этой вертикальной моды также имеет пустой список, но частная горизонтальная мода, охватывающая математическую моду, содержит некоторый клей. И наконец, мы видим основную вертикальную моду, которая охватывает все; эта мода была "`entered at line 0`" (введена на строке 0), т.е., перед тем, как файл `modes.tex` был введен; на этом самом удаленном уровне до сих пор в вертикальный список ничего не было внесено.

▸ **Упражнение 13.2**
Почему в одном из этих списков присутствует клей, а в других — нет?

▸ **Упражнение 13.3**
После результата работы команды `\showlists`, файл `modes.log` содержит результат команды `\tracingcommands`. Действительно, следующие две строки этого файла таковы

```
{math shift character $}
{horizontal mode: end-group character }}
```

поскольку `$$` на строке 7 оканчивает выделенную формулу, а это возобновляет прерванную горизонтальную моду для абзаца. Какими, как вы думаете, являются следующие три строки файла `modes.log`?

▸ **Упражнение 13.4**
Предположим, TEX генерирует документ, не выходя из вертикальной моды. Что вы можете сказать об этом документе?

▸ **Упражнение 13.5**
Некоторые моды TEX'а не могут непосредственно включать в себя другие моды. Например, выделенная математическая мода никогда прямо не заключена в горизонтальной моде, даже если формула выделяется внутри абзаца, потому что часть абзаца до формулы в горизонтальной моде всегда завершается и переносится из памяти TEX'а, прежде чем начинается обработка выделенной математической формулы. Дайте полную характеристику всем парам последовательных мод, которые могут встретиться в результате работы команды `\showlists`.

Каждый образ жизни имеет свои удобства. *Every mode of life has its conveniences.*
— SAMUEL JOHNSON, *The Idler* (1758)

[Индуистские музыканты] знают *[Hindu musicians] has eighty-four modes,*
восемьдесят четыре моды,
тридцать шесть из которых общеупотребительны, *of which thirty-six are in general use,*
и каждая из которых, по всей видимости, *and each of which, it appears,*
имеет особую выразительность и способность *has a pequliar expression and the power*
вызвать конкретное доброе чувство. *of moving some particular sentiment of affection.*
— MOUNTSTUART ELPHINSTONE, *History of India* (1841)

14

Как TEX разбивает абзацы на строки

Одна из главных обязанностей систем набора текстов — это взять длинную последовательность слов и разбить ее на строки подходящего размера. Например, каждый абзац этого руководства разбит на строки, ширина которых равна 29 pc, но автор, когда готовит рукопись, не должен заботиться о таких деталях. TEX использует для выбора таких точек разбиения интересный способ, который рассматривает абзац как единое целое; слова в конце абзаца могут даже повлиять на вид первой строки. В результате пробелы между словами насколько это возможно единообразны, и компьютер может во много раз уменьшить количество переносов слов или формул, разорванных между строками.

Эксперименты в главе 6 уже проиллюстрировали общую идею. Мы обсуждали понятие плохости ("badness"), а в трудных ситуациях сталкивались с "переполненными" и "недозаполненными" боксами. Мы также видели, что задавая различные значения параметра `\tolerance` (допуск), можно получать различные результаты. Более высокий допуск означает, что допускаются более широкие пробелы.

TEX будет искать самый лучший способ напечатать каждый абзац в соответствии с принципом наименьшей плохости. Но такая "плохость" подходит не ко всякому случаю, и если вы всецело полагаетесь на автоматическую схему, то время от времени будете сталкиваться с таким разбиением строк, которое с вашей точки зрения является не самым лучшим. Это неизбежно, потому что компьютер не понимает мотивы поведения людей (по крайней мере, пока). Поэтому может быть вы захотите сказать машине, что некоторые позиции не подходят для того, чтобы быть точками разрыва, и наоборот, иногда заставить разорвать строку в определенном месте. У TEX'а есть удобный способ избежать психологически плохого разбиения абзаца на строки, поэтому можно получать результаты высочайшего класса, просто давая машине несколько намеков. "Связки", обозначенные как ~ в начальном TEX'е, являются ключом к успешному разбиению строк. Как только вы узнаете, как их вставлять, вы выдвинетесь из ряда обычных TEXнических наборщиков в избранную группу выдающихся TEXников. И действительно, совсем не трудно приучить себя, печатая рукопись, время от времени, почти не думая, вставлять связки.

Напечатать связку — это почти то же самое, что напечатать пробел, за тем исключением, что TEX не будет на этом месте разрывать строку. Более того, нельзя оставлять никаких пробелов после ~, поскольку они будут считаться дополнительными пробелами. Если вы поместите ~ в самом конце строки входного файла, то получите более широкий промежуток, чем вам хотелось, потому что ⟨return⟩, который следует за ~, дает дополнительный пробел.

Мы уже обсуждали в главе 12, что вообще после сокращений в тексте рекомендуется печатать ~, чтобы точки в сокращениях не воспринимались как конец предложения. Знаки ~ используются также в некоторых других случаях.

- В ссылках на именованные части документа:

```
Глава~12                    Теорема~1.2
Приложение~A                Таблица~\hbox{B-8}
Рисунок~3                   Лемма  5 и~6
```

В последнем примере после слова "Лемма" знак ~ отсутствует, поскольку не будет большой беды от того, что "5 и 6" появятся в начале строки. Использование \hbox объясняется ниже).

- Между именами собственными и между частями сложной фамилии:

```
Дональд~E. Кнут                     Луи~I. Trabb~Pardo
Bartel~Leendert van~der~Waerden     Карл~XII
```

Заметим, что иногда лучше перенести имя, чем разорвать строку между словами, например: "Дон-" и "алд Е. Кнут" более приемлемо, чем "Дональд" и "Кнут". Предыдущее правило можно считать специальным случаем этого, поскольку мы можем рассматривать название "Глава 12" как составное имя; другой пример — "счетчик~X". Иногда имя может быть настолько длинным, что мы рискуем не связать его вместе, если не дать способа разбиения строки:

```
Charles Louis Xavier~Joseph de~la Vall\'ee~Poussin.
```

- Между математическими символами, присоединенными к существительным:

```
размер~$d$ширина~$w$функция~$f(x)$
строка~$s$ длиной~$l$
```

Однако, последний пример надо сравнить с

```
строка~$s$ длиной $l$~или более
```

- Между символами в ряде:

```
1,~2 или~3
$a$,~$b$ и~$c$
1,~2, \dots,~$n$
```

- Когда символ тесно связан с предлогом:

```
от~$x$
от 0 до~1
увеличить $z$ на~1
совместно с~$m$.
```

Это правило не применяется, однако, к составным объектам:

```
от $u$~ и~$v$.
```

- Когда математическое предложение выражено словами:

```
равно~$n$    меньше чем~$\epsilon$    (данное~$X$)
мод~2        модуль~$p^e$             для всех больших~$n$
```

Сравните "`равно~15`" и "`в 15~раз выше`".

- Когда случаи переименованы внутри абзаца:

```
(b)~Покажите, что $f(x)$ является
(1)~непрерывной; (2)~ограниченной.
```

Было бы прекрасно сократить все эти правила до одного или двух простых принципов, и было бы еще прекрасней, если бы эти правила могли быть автоматизированы, так чтобы можно было обойтись без них, но это потребовало бы тонкого семантического анализа. Поэтому при применении связок лучше всего воспользоваться своим собственным здравым смыслом.

Связка предохраняет TEX от разрыва строки на пробеле, но иногда хочется воспрепятствовать тому, чтобы машина разрывала строку на дефисе или тире. Это можно сделать, используя `\hbox`, потому что TEX не разбивает содержимое бокса; боксы, как только созданы, становятся неделимыми единицами. Мы уже иллюстрировали этот принцип в примере `Таблица~\hbox{B-8}`, рассмотренном ранее. Другой пример — печать номера страницы в библиографической ссылке. Если на строке окажется только "22.", то это будет выглядеть некрасиво, поэтому можно печатать "`\hbox{13--22}.`", чтобы запретить разбиение "13–22.". С другой стороны, TEX не часто выбирает разбиение строки на дефисе, поэтому вам не следует беспокоится и вставлять команду `\hbox`, если только вы не исправляете плохое разбиение, сделанное TEX'ом при предыдущем прогоне.

▸ **Упражнение 14.1**
Приведем несколько фраз, выбранных из предыдущих глав этого руководства. Как, думаете вы, автор напечатал их?

(см. главу 12)
Главы 12 и 21
строка 16 из главы 6 файла `story`
строки с 7 по 11
строки 2, 3, 4 и 5
(2) большая черная полоса
Все 128 символов изначально имеют категорию 12
буква `x` семейства 1
коэффициент f, где n в 1000 раз больше f

▸ **Упражнение 14.2**
Как бы вы напечатали фразу: "Для всех n больших чем n_0" ?

▸ **Упражнение 14.3**
А как бы вы напечатали: "упражнение 4.3.2–15" ?

▸ **Упражнение 14.4**
Почему лучше напечатать “`Глава~12`”, чем “`\hbox{Глава 12}`” ?

▸ **Упражнение 14.5**
TEX будет иногда разрывать математическую формулу после знака равенства. Как вы запретите компьютеру разрывать формулу $x = 0$?

▸ **Упражнение 14.6**
Объясните, как вы могли бы указать TEX у не делать разбиения после явных черточек и тире. (Это полезно в длинных библиографических ссылках)

Иногда вы хотите разрешить разбиение после /, как если бы это был простой дефис. Для этой цели начальный TEX позволяет вам сказать `\slash`; например, `input\slash output` дает input/output с возможным разрывом.

Если вы хотите заставить TEX сделать разрыв между строками в некоторой точке в середине абзаца, надо просто сказать `\break`. Однако, это может привести к тому, что на строке будут слишком широкие пробелы. Если вы хотите, чтобы TEX дополнил правую часть строки пробелами перед принудительным разрывом строки, чтобы следующая строка была без отступа, скажите `\hfil\break`.

Иногда бывает нужно, чтобы каждой строке на входе соответствовала строка на выходе. Одним из решений будет в конце каждой входной строки печатать `\par`, но это неприятно, поэтому начальный TEX предусматривает сокращение `\obeylines`, которое каждый конец строки рассматривает как `\par`. После того, как вы скажете `\obeylines`, вы будете получать по одной строке выхода на каждую строку входа, если только входная строка не оканчивается `%` или если она не настолько длинна, что должна разрываться. Например, удобно использовать `\obeylines` при печатании стихов. Конечно, стихи следует заключить в группу, если только вы не хотите продолжать этот “стихотворный стиль” до конца документа.

```
{\obeylines\smallskip
Эти розы неплохи,
Лилии белеют...
\quad
Мы печатаем стихи
\quad Боксами и клеем.
\smallskip}
```

▸ **Упражнение 14.7**
Объясните употребление `\quad` в этом стихотворении. Что случилось бы, если бы `\quad` в обоих местах было заменено на `\indent`?

Грубо говоря, TEX разбивает абзац на строки следующим образом. Точки разбиения вставляются между словами или после знака переноса, так что получаются строки, плохость которых не превышает текущего значения `\tolerance`. Если не существует способа вставить такие точки, фиксируется

переполненный бокс. Точки разбиения выбираются так, что абзац получается математически оптимальным, т.е. наилучшим из возможных в том смысле, что имеет не больше "дефектов", чем можно было бы получить при любой другой последовательности точек разбиения. В дефекты входят плохости отдельных строк, такие вещи, как последовательные строки, которые оканчиваются переносом, а также слишком сжатые строки, которые следуют за слишком свободными.

Но неформальное описание разбиения абзаца на строки в предыдущем абзаце является упрощением того, что может случиться в действительности. Остаток этой главы более точно объясняет детали преобразования абзаца в последовательность строк тем пользователям, которые хотят применять TEX нестандартным образом. Алгоритм TEX'а для разбиения строк оказывается достаточно всеобщим для того, чтобы справиться с удивительным разнообразием различных приложений. Фактически, это, вероятно, наиболее интересный аспект в системе TEX. Однако, каждый абзац, начиная с этого и до конца главы, предварен по меньшей мере одним знаком опасного поворота, так что вы можете пожелать изучать следующий материал постепенно, а не весь сразу.

Перед тем как быть поделенным на строки, абзац внутри TEX'а представляет собой *горизонтальный список*, т.е., последовательность элементов, которую составил TEX, когда находился в горизонтальной моде. Неформально мы уже говорили, что горизонтальный список состоит из боксов и клея, но на самом деле оказывается, что боксы и клей — это еще не все. Каждый элемент в горизонтальном списке относится к одному из следующих типов:

- бокс (символ, лигатура, линейка, h-бокс или v-бокс)
- возможный разрыв (вскоре он будет объяснен)
- "whatsit" (нечто специальное, что будет объяснено позже)
- вертикальный материал (из `\mark`, `\vadjust` или `\insert`)
- порция клея (или `\leaders`, как мы увидим позже)
- керн (что-то вроде клея, но не растяжимое и не сжимаемое)
- штраф (оценка нежелательности разрыва строки в данном месте)
- начало и конец "математики" (формулы).

Последние четыре типа элементов (клей, керн, штраф и математические элементы) называются *отпадающими* (*discardable*), поскольку при разбиении строк они могут измениться или исчезнуть. Первые четыре типа называются неотпадающими, поскольку они всегда остаются неизменными. Многое из того, что может появиться в горизонтальном списке, еще не затрагивалось в этом руководстве, и для того, чтобы понять процесс разбиения абзаца на строки, не обязательно понимать все это. Раньше или позже вы изучите, как каждый из перечисленных выше элементов может оказаться в горизонтальном списке. Если вы хотите основательно разобраться во внутренних процессах TEX'а, то используя в различных вариантах команду `\showlists`, вы всегда можете и точно увидеть, что делает TEX.

Возможный разрыв описывается тремя последовательностями символов, которые называются *доразрывными*, *послеразрывными* и *неразбитыми*. Смысл этого состоит в том, что если в данном месте произойдет разрыв строки,

доразрывная часть появится в конце текущей строки, а послеразрывная часть — в начале следующей строки. Но если разрыв строки не произойдет, то на текущей строке появится неразбитый текст. Наиболее общим образом пользователи могут указать возможный разрыв, если напишут

```
\discretionary{⟨доразрывная часть⟩}
{⟨послеразрывная часть⟩}{⟨неразбитый текст⟩}
```

где указанные три текста полностью состоят из символов, боксов и кернов. Например, TEX может расщепить слово difficult между буквами f, хотя это требует разделить лигатуру ffi на f- и следующую за ним лигатуру fi. Для этого горизонтальный список должен содержать

```
di\discretionary{f-}{fi}{ffi}cult.
```

К счастью, вам не надо самому печатать такую кутерьму — алгоритм TEX'а для переноса слов действует за кулисами, выбирая лигатуры и вставляя возможные разрывы, если это необходимо.

Наиболее общеупотребительный случай возможного разрыва — это просто возможный перенос.

```
\discretionary{-}{}{}
```

для которого в TEX'е принято сокращение `\-`. Следующий часто употребительный случай — это

```
\discretionary{}{}{}
```

("пустые возможные разрывы"), которые TEX автоматически вставляет после `-` и после каждой лигатуры, которая заканчивается знаком `-`. Поэтому в начальном TEX'е пустые возможные разрывы вставлены после дефисов и тире. (Каждый шрифт имеет соответствующий `\hyphenchar` (символ переноса), который мы для простоты можем предположить равным `-`.)

Когда TEX расщепляет слово, он просто вставляет возможные разрывы в горизонтальный список. Например, слова "`возможные переносы`", если необходим перенос, преобразуются в следующий эквивалент:

```
воз\-мож\-ные  пе\-ре\-но\-сы
```

Но TEX не применяет свой алгоритм расщепления ко всем тем словам, в которых уже явно указаны возможные разрывы, поэтому в крайнем случае вы можете явно указать расщепления, чтобы подавить действие автоматического метода TEX'а.

▸ **Упражнение 14.8**

Некоторые составные слова в немецких текстах изменяют свое написание, когда их разделяют между строками. Например "backen" становится "bak-ken", а "Bettuch" превращается в "Bett-tuch". Какие вы можете дать инструкции TEX'у, чтобы получить этот эффект?

Для того, чтобы сэкономить время, TEX пытается сначала разбить абзац на строки, не вставляя возможных переносов. Первый прогон будет успешным, если найденные точки разбиения не приводят к строкам, плохость которых превышает текущее значение `\pretolerance` (преддопуска). Если первый

прогон неудачен, используется метод из приложения H, чтобы расщепить каждое слово абзаца, вставляя в горизонтальный список возможные разрывы. Далее делается вторая попытка, использующая `\tolerance` вместо `\pretolerance`. Практика показывает, что когда строки имеют разумную ширину, как в этом руководстве, первый прогон приводит к успеху более чем в 90% случаев и что в среднем только менее чем к двум словам на абзац применяется алгоритм переноса. Но когда строки очень узки, первый прогон довольно часто оказывается неудачным. Начальный TEX по умолчанию устанавливает `\pretolerance=100` и `\tolerance=200`. Если вы сделаете `\pretolerance=10000`, первый прогон будет, по существу, всегда успешным, поэтому не будет попыток переноса (а распределение пробелов будет не очень хорошим). С другой стороны, если вы сделаете `\pretolerance=-1`, TEX пропустит первый прогон и немедленно приступит к расщеплению.

Разрывы строк могут встретиться только в определенных местах внутри горизонтального списка. Грубо говоря, они встречаются между словами и после знака переноса, но в действительности разрешены в следующих пяти случаях:

a) на клее, при условии, что этому клею непосредственно предшествует неотпадающий элемент и что он не является частью математической формулы (т.е., не расположен между включением и выключением математической моды). Разрыв "на клее" происходит на левом краю порции клея;

b) на керне, при условии, что за этим керном непосредственно следует клей и что он не является частью математической формулы;

c) на знаке отмены математической моды, за которой непосредственно следует клей;

d) на штрафах, (которые могут быть автоматически вставлены в формулу);

e) на возможном разрыве.

Заметим, что если две порции клея следуют одна за другой, вторая из них никогда не будет выбрана в качестве точки разрыва, поскольку ей предшествует клей (который неделим).

Каждой потенциальной точке разрыва приписан некоторый "штраф", который представляет собой "эстетическую цену" разрыва на этом месте. В случаях (a), (b) и (c), штраф равен нулю, в случае (d) задан явный штраф, а в случае (e) штраф равен текущему значению `\hyphenpenalty`, если доразрывный текст не пустой, или текущему значению `\exhyphenpenalty`, если доразрывный текст пустой. В начальном TEX'е `\hyphenpenalty=50` и `\exhyphenpenalty=50`.

Например, если вы говорите в некоторой точке абзаца `\penalty 100`, эта точка будет законным местом для разрыва между строками, но разрыву в этом месте будет назначен штраф равный 100. Если вы указываете `\penalty-100`, вы этим говорите TEX'у, что эта позиция является довольно хорошей точкой для разрыва, потому что отрицательный штраф реально является "премией". Строка, которая оканчивается премией, может даже иметь "заслуги" (отрицательную дефектность).

Любой штраф, который больше или равен 10000, считается настолько большим, что TEX никогда не будет разрывать строку в этом месте. И

другая крайность: любой штраф, который меньше или равен −10000, считается настолько малым, что TEX всегда сделает здесь разрыв. Макрокоманда TEX'а `\nobreak` — это просто сокращенная форма для `\penalty10000`, поскольку это запрещает разрыв строки. Связка (tie) в начальном TEX'е эквивалентна `\nobreak\␣`. В этом случае на клее, заданном при помощи `\␣`, никогда не будет разрыва, потому что клей, которому предшествует отпадающий элемент типа штрафа, никогда не будет законной точкой разрыва.

▸ **Упражнение 14.9**
Угадайте, как в начальном TEX'е определена макрокоманда `\break`?

▸ **Упражнение 14.10**
Что получается, когда вы говорите `\nobreak\break` или `\break\nobreak`?

Когда TEX встречает разрыв строки, он удаляет все следующие за разрывом отпадающие элементы до тех пор, пока не наткнется либо на неотпадающий элемент, либо на другую точку разрыва. Например, последовательность из клеев и штрафов будет удалена как одно целое, если в ней нет боксов, за исключением случая, когда оптимальная последовательность точек разбиения содержит один или более штрафов. Команды включения и выключения математической моды действуют, по существу, как керны, которые вносят пробелы, указанные при помощи `\mathsurround`. Такие пробелы исчезнут при разбиении строки, если математическая формула окажется в самом конце строки, что соответствует сформулированным выше правилам.

Плохость строки — это целое число, которое приблизительно в 100 раз больше куба пропорции, в которой клей внутри строки должен сжаться или растянуться, чтобы создать h-бокс требуемого размера. Например, если общая сжимаемость строки равна 10 pt и если клей будет спрессован всего на 9 pt, вычисленная плохость будет равна 73 (поскольку $100\times(9/10)^3 = 72.9$). Аналогично, строка, которая растягивается в два раза больше своей растяжимости, имеет плохость равную 800. Но если полученная таким образом плохость оказывается больше чем 10000, используется значение 10000. (См. обсуждение "пропорции клея" r и "порядка клея" i в главе 12. Если $i \neq 0$, то имеется бесконечная растяжимость или сжимаемость, поэтому плохость равна 0, иначе плохость приблизительно равна $\min(100r^3, 10000)$.) Переполненные боксы считаются бесконечно плохими и их по возможности избегают.

Строка, плохость которой больше или равна 13, имеет пропорцию клея, превышающую 50%. Мы называем такую строку *тесной*, если ее клей должен сжиматься, *свободной*, если ее клей должен растягиваться, и *очень свободной*, если он должен растягиваться настолько, что плохость строки больше или равна 100. Но если плохость меньше или равна 12, мы говорим, что строка *приличная*. Про две соседние строки говорят, что они визуально несовместимы, если их классификации не близки, т.е. если тесная строка следует за свободной или если приличная строка следует за очень свободной.

TEX оценивает каждую последовательность точек разбиения, суммируя *дефектность*, которая присуща каждой конкретной строке. Цель этой оценки — выбрать такие точки разбиения, которые дают наименьшую полную

дефектность. Предположим, что строка имеет плохость b и что штраф p связан с точкой разбиения в конце этой строки. Как утверждалось выше, TEX не будет рассматривать такую строку, если $p \geq 10000$ или если b превышает допуск или преддопуск. В прочих случаях дефектность такой строки определяется по формуле:

$$d = \begin{cases} (l+b)^2 + p^2, & \text{если } 0 \leq p < 10000; \\ (l+b)^2 - p^2, & \text{если } -10000 < p < 0; \\ (l+b)^2, & \text{если } p \leq -10000. \end{cases}$$

Здесь l — это текущее значение `\linepenalty`, параметра, который можно увеличить, если вы пытаетесь заставить TEX свести все абзацы к минимальному числу строк. Начальный TEX устанавливает `\linepenalty=10`. Например, строка, плохость которой равна 20 и которая заканчивается клеем, будет иметь дефектность $(10+20)^2 = 900$, если $l = 10$, поскольку клей не имеет штрафа за разрывы. Минимизация полной дефектности абзаца — это, грубо говоря, то же самое, что минимизация суммы квадратов плохостей и штрафов. Обычно это означает, что максимальная плохость каждой конкретной строки также минимизируется по всем последовательностям точек разрывов.

▸ **Упражнение 14.11**
Формула для дефектности имеет странную непоследовательность: кажется более разумным сначала определить $d = (l+b)^2 - 10000^2$ в случае, когда $p \leq -10000$. Как вы можете объяснить это очевидное противоречие?

С парами соседних строк связана дополнительная дефектность. Если две последовательные строки визуально несовместимы в том смысле, как пояснялось минутой раньше, то к d добавляется текущее значение `\adjdemerits`. Если две последовательные строки оканчиваются переносами, то добавляется значение `\doublehyphendemerits`. А если предпоследняя строка абзаца заканчивается переносом, то добавляется значение `\finalhyphendemerits`. Plain TEX задает `\adjdemerits=10000`, `\doublehyphendemerits=10000` и `\finalhyphendemerits=5000`. Дефектность выражается в единицах "квадрат плохости", поэтому чем больший эффект нужен, тем большими должны быть параметры для дефектности. Допуски и штрафы выражаются в тех же единицах, что и плохость.

Если вы установили `\tracingparagraphs=1`, то протокольный файл будет содержать итоги вычислений TEX'а при разбиении строк, так что можно наблюдать изменения, которые получаются, если играть параметрами типа `\linepenalty`, `\hyphenpenalty` и `\adjdemerits`. Данные о разбиении строк сначала выглядят жутковато, но после небольшой практики можно научиться читать их. Это фактически наилучший способ основательно разобраться в разбиении строк. Приведем трассировку, которая получается из второго абзаца файла `story` в главе 6 при `\hsize=2.5in` и `\tolerance=1000`:

```
[]\tenrm Mr. Drofnats---or ``R. J.,'' as he pre-
@\discretionary via @@0 b=0 p=50 d=2600
@@1: line 1.2- t=2600 -> @@0
ferred to be called---was hap-pi-est when
@ via @@1 b=131 p=0 d=29881
@@2: line 2.0 t=32481 -> @@1
```

```
he
@ via @@1 b=25 p=0 d=1225
@@3: line 2.3 t=3825 -> @@1
was at work type-set-ting beau-ti-ful doc-
@\discretionary via @@2 b=1 p=50 d=12621
@\discretionary via @@3 b=291 p=50 d=103101
@@4: line 3.2- t=45102 -> @@2
u-
@\discretionary via @@3 b=44 p=50 d=15416
@@5: line 3.1- t=19241 -> @@3
ments.
@\par via @@4 b=0 p=-10000 d=5100
@\par via @@5 b=0 p=-10000 d=5100
@@6: line 4.2- t=24341 -> @@5
```

Строки, которые начинаются с `@@`, представляют собой *вероятные точки разрыва*, т.е. точки разрыва, которые можно получить без того, чтобы плохость превысила допуск. Вероятные точки разрыва последовательно нумеруются начиная с `@@1`. Начало абзаца тоже считается вероятной точкой разрыва, и ее номер равен `@@0`. Строки, которые начинаются с `@`, но не с `@@`, являются кандидатами на то, чтобы стать в дальнейшем вероятной точкой разрыва. TEX при выборе окончательной точки разрыва будет отбирать наилучшего кандидата. Строки, которые не начинаются с `@`, указывают, насколько далеко продвинулся TEX в абзаце. Так, например, мы находим "`@@2: line 2.0 t=32481 -> @@1`" после "`...hap-pi-est when`" и перед "`he`". Отсюда мы узнаем, что вероятная точка разрыва `@@2` встречается в промежутке между словами "`when`" и "`he`". Запись "`line 2.0`" означает, что этот вероятный разрыв приходится на конец строки 2, и что эта строка будет очень свободной. (Окончания `.0`, `.1`, `.2`, `.3` установлены, соответственно, для очень свободных, свободных, приличных и тесных строк.) Номер строки оканчивается черточкой, если эта строка заканчивается возможным разрывом или если это конечная строка абзаца. Например, "`line 1.2-`" — это приличная строка, в которой был перенос. Запись "`t=32481`" означает, что полная дефектность от начала абзаца до `@@2` равна 32481, а "`-> @@1`" означает, что самый лучший способ дойти до `@@2` — это прийти из `@@1`. На предыдущей строке данных трассировки мы видим расчеты для набора строки от `@@1` до этой точки: плохость равна 131, штраф — 0, однако дефектность будет 29881 . Аналогично, точка разрыва `@@3` представляет собой альтернативу для второй строки абзаца, которая получается при разрыве между "`he`" и "`was`". Эта точка разрыва делает вторую строку тесной и дефектность после добавления дефектности строки 1 получается только 3825, так что ясно, что точка `@@3` будет работать намного лучше, чем `@@2`. Однако, следующая вероятная точка разрыва (`@@4`) встречается после "`doc-`", и дефектность строки от `@@2` до `@@4` равна только 12621, в то время как дефектность строки от `@@3` до `@@4` огромна — 103101. Поэтому из `@@0` в `@@4` лучше всего добираться через `@@2`. Если взглянуть на дефектности как на расстояния, TEX ищет "наикратчайший путь" от `@@0` до каждой вероятной точки разрыва (используя вариант хорошо известного алгоритма для наикратчайшего пути в ориентированном графе без петель). Наконец, окончание абзаца приходится на точку `@@6`, и наикратчайший путь от `@@0` до `@@6` представляет собой наилучшую последовательность точек разрыва. Сле-

дуя в обратном направлении от `@@6`, мы заключаем, что наилучшее разбиение в этом конкретном абзаце проходит через `@@5`, `@@3` и `@@1`.

▸ **Упражнение 14.12**
Объясните, почему на отрезке от `@@1` до `@@2` дефектность равна 29881, а на отрезке от `@@2` до `@@4` она равна 12621.

Если в такой трассировке появляется `b=*` , это означает, что в качестве точки разрыва выбрана недопустимая точка разрыва, поскольку не существовало допустимой альтернативы.

Мы еще не обсуждали специальный фокус, который позволяет последней строке абзаца быть короче, чем остальные. Перед тем, как TEX начинает выбирать точки разрыва, он делает две важные вещи. (1) Если конечный элемент в текущем горизонтальном списке —это клей, то он отпадает. (Дело в том, что пробел часто ставится прямо перед `\par` или перед `$$`, и такой пробел не должен быть частью абзаца.) (2) Еще три элемента ставятся в конце текущего горизонтального списка: `\penalty10000` (что запрещает разрыв строки), `\hskip\parfillskip` (что добавляет "завершающий клей" к абзацу) и `\penalty-10000` (что принуждает к разрыву). Начальный TEX устанавливает `\parfillskip=0pt plus1fil`, так что последняя строка каждого абзаца будет, если это необходимо, дополнена пробелами, но в специальных приложениях `\parfillskip` присваиваются другие значения. Например, этот абзац заканчивается впритык к правым полям, поскольку он был напечатан с `\parfillskip=0pt`. Чтобы получить такую возможность, автор не должен переписывать что-нибудь в тексте, поскольку длинный абзац обладает настолько большой гибкостью, что можно заставить сделать разрыв строки почти в любой точке. Вы можете позабавляться с абзацами, поскольку алгоритм для разбиения строк иногда бывает ясновидящим. Только пишите абзацы, которые будут достаточно длинными.

▸ **Упражнение 14.13**
Ben User решил в конце абзаца сказать `\hfilneg\par` предположив, что отрицательная растяжимость `\hfilneg` будет отменена при помощи `\parfillskip` начального TEX'а. Почему эта светлая мысль не сработала?

▸ **Упражнение 14.14**
Как вы можете задать `\parfillskip`, чтобы последняя строка абзаца имела справа столько пробелов, каков отступ на первой строке слева?

▸ **Упражнение 14.15**
Поскольку TEX перед тем, как принять решение о разрывах строк, прочитывает абзац целиком, то память компьютера может быть переполнена, если вы печатаете труды какого-нибудь философа или писателя-модерниста, который пишет длиннющими абзацами по 200 строк. Попробуйте предложить способ работы с такими авторами.

У TEX'а есть два параметра `\leftskip` и `\rightskip`, которые задают клей, вставляемый по краям каждой строки абзаца. Этот клей учитывается, когда вычисляются плохости и дефектность. Начальный TEX обычно принимает `\leftskip` и `\rightskip` равными 0, но имеет макрокоманду `\narrower` (у́же), которая увеличивает оба эти значения на текущую величину `\parindent`.

Вы можете воспользоваться `\narrower`, когда цитируете длинный отрывок из книги:

```
{\narrower\smallskip\noindent
У этого абзаца будут более узкие строки, чем
вокруг, потому что он использует команду
narrower начального \TeX'a.
Прежние  поля будут восстановлены после того, как
группа закончится.\smallskip}
```

(Попробуйте сделать это). Второй `\smallskip` в этом примере заканчивает абзац. Важно закончить абзац перед окончанием группы, иначе действие `\narrower` исчезнет еще до того, как TEX начнет выбирать разрывы строк.

▸ **Упражнение 14.16**

Когда абзац целиком печатается курсивом или наклонным шрифтом , на странице по отношению к другим абзацам может появиться отступ. Объясните, как можно воспользоваться параметрами `\leftskip` и `\rightskip`, чтобы подвинуть строки абзаца влево на 1 pt.

▸ **Упражнение 14.17**

Макрокоманды `\centerline`, `\leftline`, `\rightline` и `\line` начального TEX'a не учитывают `\leftskip` и `\rightskip`. Как можно заставить их делать это?

Если вы подозреваете, что `\raggedright` устанавливается некоторыми соответствующими манипуляциями с `\rightskip`, то вы правы. Но будьте внимательны: кто-нибудь может задать `\rightskip=0pt plus1fil`, и каждая строка справа будет дополнена пробелами. Но это не самый лучший способ создания рваных правых полей, поскольку бесконечная растяжимость будет присваивать нулевую плохость строкам, которые очень коротки. Для того, чтобы качественно задать неровный правый край, есть такой фокус. Надо установить `\rightskip` так, что он будет растягиваться достаточно, чтобы сделать возможным разбиение строк, но не слишком сильно, потому что короткие строки должны рассматриваться как плохие. Более того, пробелы между словами должны быть фиксированными, т.к. они не растягиваются и не сжимаются. (См. определение `\raggedright` в приложении B.) Можно также позволить небольшую изменчивость междустрочного клея, так чтобы правые поля не были слишком рваными, но абзац все еще имел неформальный вид.

TEX смотрит на параметры, которые влияют на разрыв строк, только когда производит этот разрыв. Например, вы не можете в середине абзаца пытаться изменить `\hyphenpenalty` , если хотите, чтобы TEX оштрафовал перенос одного слова больше, чем другого. Используемые значения `\hyphenpenalty`, `\rightskip`, `\hsize` и так далее — это текущие значения этих величин в конце абзаца. С другой стороны, ширина отступа, которую вы получаете в начале абзаца неявно или при помощи `\indent`, определяется значением `\parindent` в то время, когда отступ вносится в текущий горизонтальный список, а совсем не его значением в конце абзаца. Аналогично, штрафы, которые вставляются в математические формулы внутри абзаца, основываются на тех значениях `\binoppenalty` и `\relpenalty`, которые являются текущими в конце каждой отдельной формулы.

Приложение D содержит пример, который показывает, как в одном и том же абзаце получить поля, рваные как справа, так и слева, не используя `\leftskip` или `\rightskip`.

Есть гораздо более общий способ управлять длиной строки, если простое изменение параметров `\leftskip` и `\rightskip` для ваших целей оказывается не достаточно гибким. Например, в этом абзаце вырезано полукруглое углубление для того, чтобы подготовить место для круглой иллюстрации, которая содержит бессмертные слова Галилея о кругах. Все разрывы строк в этом абзаце были найдены с помощью алгоритма TEX'а для разбиения абзаца на строки. Вы можете задать в сущности произвольную форму абзаца, указав `\parshape=`⟨число⟩, где ⟨число⟩ — это положительное целое n, за которым следует спецификация $2n$ ⟨размер⟩. Вообще, `\parshape=`$n\ i_1\ l_1\ i_2\ l_2\ \ldots\ i_n\ l_n$ задает абзац, у которого n строк будут иметь длины соответственно l_1, l_2, …, l_n, а также будут отступать от левого поля на величины, соответственно, i_1, i_2, …, i_n. Если у абзаца меньше n строк, то дополнительные спецификации будут игнорироваться. Если строк больше чем n, то будет бесконечно повторяться спецификация n-ой строки. Можно отменить действие предыдущего указания `\parshape`, сказав `\parshape=0`.

Площадь круга есть среднее геометрическое от площадей двух правильных и подобных многоугольников, из которых один описан вокруг круга, а другой изопериметричен ему. К тому же, площадь круга меньше площади любого вписанного многоугольника. И далее, для описанных многоугольников тот, который имеет больше сторон, имеет большую площадь; но напротив, изопериметрический многоугольник с большим числом сторон является большим по площади.
[Galileo, 1638]

▸ **Упражнение 14.18**
Напечатайте следующую цитату Паскаля в форме треугольника: "I turn, in the following treatises, to various uses of those triangles whose generator is unity. But I leave out many more than I include; it is extraordinary how fertile in properties this triangle is. Everyone can try his hand."

Вам, вероятно, не часто потребуется необычная форма абзаца. Но есть специальный случай, который встречается довольно часто, поэтому TEX предусматривает специальные параметры, которые называются `\hangindent` и `\hangafter`. Команда `\hangindent=`⟨размер⟩ указывает так называемый подвешенный отступ, а команда `\hangafter=`⟨число⟩ указывает продолжительность этого отступа. Пусть x и n — значения `\hangindent` и `\hangafter`, и пусть h — значение `\hsize`. Тогда если $n \geq 0$, то подвешенный отступ встретится на строках абзаца $n+1$, $n+2$, …, а если $n < 0$, он появится на строках 1, 2, …, $|n|$. Подвешенный отступ означает, что строки будут шириной $h - |x|$ вместо их обычной ширины h. Если $x \geq 0$, строки будут отступать от левого поля, иначе — от правого поля. Например, абзацы этого руководства, которые помечены знаками "опасного поворота", имеют подвешенный отступ равный 3 pc, который продолжается на двух строках. Это задано при помощи `\hangindent=3pc` и `\hangafter=-2`.

Начальный TEX использует подвешенный отступ в своей макрокоманде `\item`, производящей абзац, каждая строка которого имеет один и тот же отступ, равный обычному `\indent`. Более того, `\item` имеет параметр, который помещается в позицию отступа первой строки. Другая макрокоманда `\itemitem` делает то же самое, но с двойным отступом. Например, предположим, вы вводите

```
\item{1.} Это первый из нескольких перенумерованных случаев с
подвешенным отступом,
примененным к целому абзацу.
\itemitem{a)} Это первый  подслучай.\cr
\itemitem{b)} Это второй подслучай. Заметим, что подслучаи имеют в
два раза больший отступ.\cr
\item{2.} Второй случай аналогичен.\cr
```

Тогда вы получите следующий результат:

1. Это первый из нескольких перенумерованных случаев с подвешенным отступом, примененным к целому абзацу.
    a) Это первый подслучай
    b) Это второй подслучай. Заметим, что подслучаи имеют в два раза больший отступ.
2. Второй случай аналогичен.

(Отступы в начальном TEX'е меньше тех, которые показаны здесь. Приложение B говорит, что `\parindent=20pt`, но в этом руководстве было установлено `\parindent=36pt`.) Обычно принято перед и после группы нумерованных абзацев ставить `\medskip`, а перед заключающим примечанием, которое относится ко всем случаям, ставить `\noindent`. Перед `\item` или `\itemitem` не нужны пустые строки, поскольку эти команды начинаются с `\par`.

▸ **Упражнение 14.19**
Предположим, что один из перенумерованных случаев состоит из двух или более абзацев. Как вы можете, использовать `\item`, чтобы получить подвешенный отступ на последующих абзацах?

▸ **Упражнение 14.20**
Объясните, как сделать "простреленный пункт", в котором вместо "1" помещен знак •

▸ **Упражнение 14.21**
Макрокоманда `\item` не делает отступ от правого поля. Как бы вы могли сделать отступ с обеих сторон?

▸ **Упражнение 14.22**
Объясните, как можно задать подвешенный отступ, равный −2 em (т.е., строки должны выдаваться на левые поля) после первых двух строк абзаца.

Если заданы как `\parshape`, так и подвешенный отступ, то приоритет имеет `\parshape` , а `\hangindent` игнорируется. Если же `\parshape=0`, `\hangindent=0pt` и `\hangafter=1`, то получается обычная форма абзаца, в которой каждая строка имеет ширину `\hsize`. TEX автоматически восстанавливает эти обычные значения в конце каждого абзаца и (при помощи локальных определений) когда входит во внутреннюю вертикальную моду. Например, подвешенный отступ, который мог быть снаружи конструкции `\vbox`, не будет сделан внутри этого v-бокса, если только вы не запрашиваете его изнутри.

Упражнение 14.23

Предположим, вы хотите оставить место на правых полях для прямоугольной иллюстрации, которая занимает 15 строк, и предполагаете, что потребуется три абзаца текста до конца этой иллюстрации. Предложите хороший способ сделать это без проб и ошибок, помня то, что TEX восстанавливает подвешенный отступ.

Если выделенные уравнения встречаются в абзаце, который имеет нестандартную форму, TEX всегда предполагает, что выделение занимает в точности три строки. Например, длина абзаца, у которого четыре строки текста, затем выделение, затем еще две строки текста, считается равной 4+3+2 = 9 строк. Выделенное уравнение будет напечатано с отступом и центрировано с помощью информации о форме абзаца, присущей строке 6.

TEX имеет внутреннюю целую переменную `\prevgraf`, в которой записано число строк самого последнего абзаца, который завершен либо полностью, либо частично. Вы можете использовать `\prevgraf` как ⟨число⟩, а также присвоить `\prevgraf` любое желаемое неотрицательноеце значение, если хотите убедить TEX в том, что он находится в некоторой конкретной части текущей формы абзаца. Например, давайте рассмотрим снова абзац, который содержит четыре строки плюс выделение плюс еще две строки. Когда TEX начинает абзац, он устанавливает `\prevgraf=0`, когда он начинает выделение, то `\prevgraf` будет равно 4, когда выделение заканчивается, `\prevgraf` равно 7, а когда заканчивается абзац, `\prevgraf` будет равно 9. Если нужно сделать выделение на одну строку толще обычного, можно установить `\prevgraf=8` перед двумя конечными строками. Тогда TEX будет считать, что создается абзац из 10 строк. Значение `\prevgraf` действует на разбиение строк только тогда, когда TEX имеет дело с нестандартными величинами `\parshape` или `\hangindent`.

Упражнение 14.24

Выполните упражнение 14.23, используя `\prevgraf`.

Вы, вероятно, уже убедились, что алгоритм разбиения строк TEX'а содержит много звона и свиста, вероятно даже слишком много. Но есть еще одна характеристика, называемая "свободность" (looseness). Она иногда может понадобиться, когда начисто отделываются страницы книги. Если вы установите `\looseness=1`, TEX попытается сделать текущий абзац на одну строку длиннее его оптимальной длины, при условии, что можно выбрать такие точки разрыва, при которых не превышается допуск, заданный для плохости отдельных строк. Аналогично, если вы установите `\looseness=2`, TEX попытается сделать абзац на две строки длиннеее, а `\looseness=-1` заказывает попытку сделать его короче. Общая идея такова: TEX сначала пытается найти точки разрыва, как обычно. Затем, если оптимальные точки разрыва дают n строк и если текущие значения `\looseness` равны l, TEX будет выбирать окончательные точки разрыва так, чтобы сделать число строк насколько можно близкими к $n + l$, не превышая текущего допуска. К тому же окончательные точки разрыва будут иметь наименьшую полную дефектность из всех способов достижения того же числа строк.

Например, вы можете установить `\looseness=1`, если хотите избежать "висячей" или "вдовьей" строки на странице, которая не имеет доста-

точно гибкого клея, или если хотите, чтобы общее число строк в каком-либо двухколоночном документе было одинаковым. Обычно лучше всего выбрать абзац, который уже красиво "заполнен", т.е. тот, у которого последняя строка не содержит слишком много пустого места, поскольку такой абзац вообще может быть ослаблен без особого ущерба. Можно также между двумя последними словами вставить связку, чтобы ослабленная версия не заканчивалась только одним "висячим словом" на строке. Эта связка заметет вам следы, так что будет трудно обнаружить тот факт, что вы вмешивались в расстановку пробелов. С другой стороны, TEX может взять почти любой достаточно длинный абзац и немножко растянуть его без значительного ущерба. Этот абзац как раз на одну строку свободнее своей оптимальной длины.

TEX сбрасывает свободность на нуль в то же время, когда восстанавливает `\hangindent`, `\hangafter` и `\parshape`.

Упражнение 14.25
Объясните, что будет делать TEX, если установить `\looseness=-1000`.

Непосредственно перед переключением в горизонтальную моду для просмотра абзаца TEX вставляет в вертикальный список, который будет содержать абзац, клей, заданный при помощи `\parskip`, если этот вертикальный список до сих пор не пуст. Например, `\parskip=3pt` укажет, что между абзацами нужно поместить дополнительный промежуток величиной в 3 pt. Начальный TEX устанавливает `\parskip=0pt plus1pt`. Это дает маленькое растяжение , но не дает дополнительного пробела.

После того, как завершено разбиение строк, TEX присоединяет строки к вертикальному списку, который охватывает текущий абзац, вставляя междустрочный клей, как это объяснялось в главе 12. Этот междустрочный клей будет зависеть от значений `\baselineskip`, `\lineskip` и `\lineskiplimit`, которые действуют в это время. TEX также вставляет в вертикальный список штрафы перед каждой порцией междустрочного клея для того, чтобы управлять разбиением на страницы, которое надо будет делать позже. Например, специальным штрафом будет оценено разбиение страниц между первыми двумя строками абзаца или перед последней строкой, так что "висячие" строки, которые отделены от остатка абзаца, не появятся в одиночестве на странице, если только альтернативные варианты не хуже.

Вот как вычисляются междустрочные штрафы: TEX только что выбрал точки разрыва для некоторого абзаца или для некоторой части абзаца, которая предшествует выделенному уравнению и при этом сформировано n строк. Штраф между строками j и $j+1$, где j в пределах $1 \le j < n$, равен значению `\interlinepenalty` плюс дополнительные расходы, сделанные в специальных случаях: добавляется `\clubpenalty`, если $j = 1$, т.е., сразу после первой строки, затем `\displaywidowpenalty` или `\widowpenalty`, если $j = n - 1$, т.е., непосредственно перед последней строкой, в зависимости от того, следует или нет за текущими строками выделенное уравнение, и, наконец, добавляется `\brokenpenalty`, если j-ая строка заканчивается возможным разрывом. (Начальный TEX устанавливает значение `\clubpenalty=150`, `\widowpenalty=150`, `\displaywidowpenalty=50` и `\brokenpenalty=100`; значение `\interlinepenalty` обычно равно нулю, но оно

увеличивается до 100 внутри сноски, так что длинные сноски стараются не разрываться между страницами.)

▸ **Упражнение 14.26**
Рассмотрите абзац из пяти строк, в котором пятая и четвертая строки оканчиваются дефисами. Какие штрафы ставит начальный TEX между строками?

▸ **Упражнение 14.27**
Какой штраф будет между строками в двухстрочном абзаце?

Если вы внутри абзаца говорите `\vadjust{`⟨вертикальный список⟩`}`, TEX вставляет указанный внутренний вертикальный список в вертикальный список, который охватывает абзац, непосредственно после строки, в которой содержится `\vadjust`. Например, можно сказать `\vadjust{\kern1pt}` для того, чтобы увеличить величину пробела между строками абзаца, если без этого строки получились слишком близкими друг к другу. (Автор сделал это на этой строке, чтобы проиллюстрировать, что получилось.) Если вы хотите быть уверенными, что разбиение произойдет непосредственно после некоторой строки, то можете где-нибудь на этой строке сказать `\vadjust{\eject}` .

Позже будут обсуждаться команды `\insert` и `\mark`, которые относятся к механизму построения страниц TEX'а. Если такие команды появляются внутри абзаца, они удаляются из содержащей их горизонтальной строки и помещаются в вертикальный список вместе с другим вертикальным материалом из команд `\vadjust`, которые могли встретиться. В окончательном вертикальном списке каждая горизонтальная строка текста является горизонтальным боксом, непосредственно перед которым расположен междустрочный клей, а за которым следует вертикальный материал, "переселенный" из этой строки (если есть несколько случаев вертикального материала, то сохраняется порядок слева направо), затем следует междустрочный штраф, если он не равен нулю. Вставленный вертикальный материал не влияет на междустрочный клей.

▸ **Упражнение 14.28**
Сконструируйте макрокоманду `\marginalstar`, которая может быть применена в любом месте абзаца. Она должна использовать `\vadjust` для того, чтобы поместить звездочку слева от строки, на которой встречается `\marginalstar`.

Когда TEX входит в горизонтальную моду, он прерывает обычный просмотр, чтобы прочитать элементы, которые были ранее определены командой `\everypar={`⟨список элементов⟩`}`. Например, предположим вы сказали `\everypar={A}`. Если вы введете "`B`" в вертикальной моде, TEX переключится в горизонтальную моду (после внесения клея `\parskip` в текущую страницу) и начнет горизонтальный список вставкой пустого бокса шириной `\parindent`. Затем TEX прочтет "`AB`", поскольку он читает элементы `\everypar` перед возвращением к "`B`", которая включила новый абзац. Конечно, это не очень полезная иллюстрация для команды `\everypar`, но если вы дадите простор своему воображению, то придумаете ей лучшее применение.

▸ **Упражнение 14.29**
Определите макрокоманду `\insertbullets`, используя `\everypar`: все абзацы вида `{\insertbullets ...\par}` должны иметь как часть своего отступа символ •.


Абзац без строк формируется, если вы говорите `\noindent\par`. Если `\everypar` равно нулю, такой абзац не заносит в текущий вертикальный список ничего, кроме клея `\parskip`.


▸ **Упражнение 14.30**
Угадайте, что случится, если вы скажете `\noindent$$...$$ \par`.


Опыт показывает, что алгоритм TEX'а для разбиения строк может быть использован в удивительном разнообразии задач. Например, приведем одну из возможностей. Статьи, опубликованные в *Mathematical Reviews*, обычно сопровождаются именем автора и адресом. Эта информация печатается справа, т.е. вплотную к правым полям. Если есть достаточно места, чтобы поместить имя и адрес в правой части последней строки абзаца, издатель может сэкономить место, и в то же время результат будет выглядеть лучше, так как на странице не будет странных брешей.

> Вот случай, когда имя и адрес прекрасно подогнаны к статье. *A. Reviewer* (Ann Arbor, Mich.)

> Но иногда должна быть добавлена особая строка.
> *N. Bourbaki* (Paris)

Давайте предположим, что имя автора должно быть отделено от текста статьи по меньшей мере промежутком в 2 em, если они встречаются на одной строке. Мы бы хотели сконструировать такую макрокоманду, чтобы примеры, показанные выше, могли быть напечатаны во входном файле следующим образом:

```
... к статье. \signed A. Reviewer (Ann Arbor, Mich.)
... добавляется особая строка. \signed N. Bourbaki (Paris)
```

Далее приводится способ решения этой задачи:

```
\def\signed #1 (#2){{\unskip\nobreak\hfil\penalty50
  \hskip2em\hbox{}\nobreak\hfil\sl#1\/ \rm(#2)
  \parfillskip=0pt \finalhyphendemerits=0 \par}}
```

Если разбиение строки происходит на `\penalty50`, `\hskip2em` исчезнет, и в строке окажется пустой `\hbox` со следующим за ним `\hfil`-клеем. Это даст две строки, плохость которых равна нулю. Первая из этих строк оценивается штрафом, равным 50. Но если нет разрыва строки с `\penalty50`, то между статьей и именем будет клей 2 em plus 2 fil . Это дает одну строку с плохостью, равной нулю. TEX испробует обе альтернативы, чтобы посмотреть, какая из них приведет к наименьшей полной дефектности. Обычно предпочтительнее однострочное решение, если оно возможно.


▸ **Упражнение 14.31**
Объясните, что произойдет, если в макрокоманде `\signed` опустить `\hbox{}`


▸ **Упражнение 14.32**
Зачем макрокоманда `\signed` говорит `\finalhyphendemerits=0`?

Упражнение 14.33

В одном из абзацев этой главы автор использовал `\break`, чтобы заставить сделать разбиение строк в специфическом месте. В результате третья строка этого абзаца оказалась разреженной. Объясните, почему все дополнительные пробелы оказались в третьей строке, вместо того, чтобы быть справедливо распределенными между первыми тремя строками.

Упражнение 14.34

Придумайте макрокоманду `\raggedcenter` (аналогичную `\raggedright`) которая разделяет слова абзаца на как можно меньшее число строк приблизительно равного размера и центрирует каждую строку. Переносов, если возможно, надо избежать.

Когда автор протестует против [переноса],
его надо попросить добавить, удалить
или заменить одно или несколько слов,
чтобы предотвратить разбивку.

When the author object to [hyphenation]
he should be asked to add or cancel
or substitude a word or words that will
prevent the breakage.

Авторы, которые настаивают на всегда одинаковых пробелах с всегда красивым разбиением, не вполне понимают жесткость шрифтов.

Authors who insist on even spacing always, with sightly divisions always, do not clearly understand the rigidity of types.

— T. L. DE VINNE, *Correct Composition* (1901)

При переиздании своих трудов,
когда [Вильям Моррис] находил строку,
которая была неуклюже выровнена, он менял
формулировку единственно ради того,
чтобы при печати она хорошо выглядела.

In reprinting his own works,
whenever [William Morris] found a line
that justified awkwardly, he altered
the wording solely for the sake
of making it look well in print.

Когда ко мне присылали корректуру
с двумя-тремя строками, которые
были настолько просторны, что получалась
серая лента поперек страницы,
я обычно переписывал часть текста так,
чтобы лучше заполнить строки.
Но, к сожалению, я должен сказать,
что моя цель была настолько не понята,
что наборщик портил весь остальной абзац
вместо того, чтобы исправить
свою прежнюю плохую работу.

When the proof has been sent me
with two or three lines so widely
spaced as to make
a grey band across the page,
I have often rewritten the passage so as
to fill up the lines better;
but I am sorry to say that
my object has generally been so little
understood that the compositor has spoilt
all the rest of the paragraf instead
of mending his former bad work.

— GEORGE BERNARD SHAW , in *The Dolphin* (1940)

15

Как TEX собирает строки в страницы

TEX пытается выбрать подходящее место, чтобы разделить ваш документ на отдельные страницы, и технология, с помощью которой он делает это, достаточно хорошо работает. Но задача создания страниц намного труднее, чем задача разбиения строк, которую мы рассматривали в предыдущей главе, потому что страницы часто имеют намного меньшую гибкость, чем строки. Если вертикальный клей на странице имеет малую растяжимость или сжимаемость, то у TEX'а обычно нет выбора, где начать новую страницу. Наоборот, если у клея слишком большая гибкость, результат будет выглядеть плохо, потому что разные страницы будут слишком неодинаковыми. Поэтому, если вас слишком заботит внешний вид страниц, можно ожидать, что вам придется несколько раз переписывать рукопись, пока вы не достигнете подходящего баланса, или понадобится повозиться с `\looseness`, как описано в главе 14. Ни одна автоматизированная система не сможет сделать это так хорошо, как это сделаете вы.

Математические документы, которые обычно содержат много выделенных уравнений, имеют в этом отношении преимущество, потому что клей, который окружает выделенные уравнения, обычно очень гибкий. TEX также получает ценное место для маневрирования, когда вы между абзацами используете `\smallskip`, `\medskip` или `\bigskip` . Например, рассмотрим страницу, которая содержит примерно дюжину упражнений и предположим, что между упражнениями добавлен промежуток в 3 pt и что этот промежуток может растягиваться до 4 pt или сжиматься до 2 pt. Тогда есть возможность втиснуть на страницу дополнительную строку и перейти на новую страницу, передвинув одну строку, чтобы обойтись без разбиения упражнений между страницами. Аналогично, можно использовать гибкий клей в специальных изданиях, таких как членские списки или справочник телефонной компании, так чтобы отдельные записи в них не надо было бы разрывать между колонками или страницами и, вдобавок, чтобы каждая колонка была бы одинаковой высоты.

Для повседневной работы вас, вероятно, устроит автоматический метод TEX'а для разбиения страниц. А если окажется, что этот метод дает неприятный результат, вы можете заставить машину разорвать страницу на понравившемся вам месте, напечатав `\eject`. Но будьте внимательны: `\eject` указывает TEX'у растянуть страницу, если это необходимо, так что верхние и нижние базовые линии будут согласовываться с такими же базовыми линиями других страниц. Если вы хотите вывести укороченную страницу, дополненную до конца пробелами, напечатайте вместо этого `\vfill\eject`.

Если вы говорите `\eject` в середине абзаца, прежде всего будет закончен абзац, как если бы вы напечатали `\par\eject`. Но в главе 14 упоминалось, что можно сказать `\vadjust{\eject}` в середине абзаца, если нужно, чтобы страница разорвалась после той строки, которая содержит текущую позицию, когда целый абзац окончательно разбивается на строки; остаток абзаца перейдет на следующую страницу.

Чтобы помешать разрыву страницы, можно сказать `\nobreak` в вертикальной моде, так же как `\nobreak` в горизонтальной моде мешает разрыву между строками. Например, можно между заголовком и первой строкой текста раздела сказать `\nobreak`. Но `\nobreak` не отменяет действие других команд типа `\eject`, которые указывают TEX'у сделать разрыв. Он только задерживает разрыв до ближайшего соседнего клея. Вы должны познакомиться с правилами TEX'а для распределения строк по страницам, если хотите управлять им. Остаток этой главы посвящен сокровенным деталям разбиения страниц.

TEX разбивает список строк на страницы, вычисляя плохости и штрафы более или менее так же, как он это делает, когда разбивает абзац на строки. Но страницы создаются один раз и убираются из памяти TEX'а — нельзя заглянуть вперед и посмотреть, как формирование одной страницы действует на следующую. Другими словами, TEX использует метод для нахождения оптимальных точек разрыва для строк в целом абзаце, но не пытается найти оптимальные точки разрыва для страниц в целом документе. У компьютера нет достаточной емкости быстродействующей памяти, чтобы запомнить содержимое нескольких страниц, так что TEX выбирает разбиение каждой страницы насколько может хорошо, производя "локальную", а не "глобальную" оптимизацию.

Давайте посмотрим теперь на детали процесса создания страницы. Все, что вы заносите на страницу документа, помещается в *основной вертикальный список*, представляющий собой последовательность элементов, которые TEX накапливает в вертикальной моде. Каждый элемент в вертикальном списке относится к одному из следующих типов:

- бокс (h-бокс, v-бокс или линейка);
- "whatsit" (нечто специальное, что будет объяснено позже);
- метка (специальная штука, которая будет объяснена позже);
- вставка (до нее дойдем позднее);
- порция клея (или `\leaders`, как мы позднее увидим);
- керн (что-то вроде клея, но не растягивается и не сжимается);
- штраф (представляет нежелательность разбиения в этом месте).

Последние три вида (клеи, керны и штрафы) называются отпадающими элементами по той же причине, по которой мы называем их отпадающими в горизонтальном списке. Можно сравнить эти спецификации с аналогичными правилами для горизонтального случая, указанными в главе 14. Оказывается, что вертикальные списки точно такие же, как горизонтальные, за исключением того, что в вертикальных списках не могут встречаться символьные боксы, возможные разрывы, элементы `\vadjust` и математические переключатели. Глава 12 демонстрирует типичный вертикальный список во внутреннем представлении TEX'а.

Разбиение страниц может произойти только в определенных местах вертикального списка. Допустимые точки разрыва точно такие же, как и в горизонтальном случае, а именно:

a) на клее, при условии, что этому клею непосредственно предшествует неотпадающий элемент (т.е., бокс, whatsit, метка или вставка);

b) на керне, при условии, если за этим керном немедленно следует клей;

c) на штрафе (который мог быть автоматически вставлен в абзац).

В вертикальном списке между боксами обычно автоматически вставляется междустрочный клей, как это объяснялось в главе 12, так что место между боксами часто является подходящей точкой разрыва.

Как и в горизонтальных списках, каждая потенциальная точка разрыва связана со штрафом, который высок для нежелательных точек разрыва и отрицателен для желательных. Штраф равен нулю для разрывов на клее и керне, так что он ненулевой только на явных штрафных разрывах. Если вы говорите `\penalty-100` между двумя абзацами, вы указываете, что TEX должен в этом месте попытаться сделать разрыв, потому что штраф отрицателен, а премия в 100 баллов за разрыв в этом месте по существу компенсирует 100 единиц плохости, которые обязательно появились бы, если бы разрыв был сделан в этой точке. Штраф в 10000 или более единиц настолько велик, что запрещает разрыв, а штраф в -10000 или меньше единиц так мал, что принуждает к разрыву.

Начальный TEX содержит некоторые команды, которые помогают управлять разрывом строк. Например, `\smallbreak`, `\medbreak` и `\bigbreak` задают все более и более желательные места для разрыва, имеющие, соответственно, штрафы -50, -100 и -200. Более того, если разрыв не сделан, они вставят, соответственно, промежутки `\smallskip`, `\medskip` и `\bigskip`. Однако, `\smallbreak`, `\medbreak` и `\bigbreak` не увеличивают сверх необходимости существующий клей. Например, если вы говорите `\smallbreak` сразу после выделенного уравнения, то не получите пробел `\smallskip` вдобавок к клею, который уже следует за этим уравнением. Поэтому эти команды удобно использовать до или после условий теорем в формате для математических работ. В данном руководстве автор применил макрокоманду, которая ставит `\medbreak` до и после каждого "опасного" абзаца; `\medbreak\medbreak` эквивалентно одному `\medbreak`, поэтому, когда один абзац оканчивается, а другой начинается, `\medbreak` не срабатывает дважды.

Макрокоманда `\goodbreak` является сокращением для `\par\penalty-500`. Ее удобно вставлять в рукопись при корректировании, если надо растянуть немного какую-нибудь страницу, чтобы улучшить следующую. Позднее, если вы делаете другое изменение, такое, что команда `\goodbreak` появляется на нижней части страницы, она не будет действовать; таким образом, она не настолько сильнодействующая, как `\eject`.

Наиболее интересная макрокоманда, которую TEX может использовать для создания страницы, называется `\filbreak`. Грубо говоря, это означает: "Прерви страницу здесь и заполни ее до конца пробелами, если нет места для следующего материала, за которым следует `\filbreak`". Таким образом, если вы ставите `\filbreak` в конце каждого абзаца и абзацы не слишком длинны, все разрывы страниц окажутся между абзацами, и на каждой странице будет помещаться столько абзацев, сколько возможно. Точным значением `\filbreak` в соответствии с приложением B является

```
\vfil\penalty-200\vfilneg
```

Эта простая комбинация примитивов TEX'а приводит к нужному результату. Если разрыв был сделан на `\penalty-200`, предшествующий `\vfil` дополнит низ страницы пробелами, и `\vfilneg` после разрыва будет отменен. Но если разрыв не на штрафе, то `\vfil` и `\vfilneg` компенсируют друг друга.

Начальный TEX также имеет команду `\raggedbottom`, которая является вертикальным аналогом `\raggedright`. Она допускает небольшую изменчивость в нижних полях различных страниц для того, чтобы сделать другие пробелы единообразными.

Мы видели в главе 14, что точки разрыва для абзацев выбираются вычислением "дефектностей" для каждой строки и суммированием их по всем строкам. Ситуация для страниц более проста, поскольку каждая страница рассматривается отдельно. TEX вычисляет "стоимость" (cost) страницы, используя следующие формулы:

$$c = \begin{cases} p, & \text{если } b < \infty \text{ и } p \le -10000 \text{ и } q < 10000; \\ b + p + q, & \text{если } b < 10000 \text{ и } -10000 < p < 10000 \text{ и } q < 10000; \\ 100000, & \text{если } b \ge 10000 \text{ и } -10000 < p < 10000 \text{ и } q < 10000; \\ \infty, & \text{если } (b = \infty \text{ или } q \ge 10000) \text{ и } p < 10000. \end{cases}$$

Здесь b — плохость страницы, если бы разрыв был выбран здесь; p — штраф, связанный с текущей точкой разрыва, а q равно `\insertpenalties` — сумме всех штрафов для расщепленных вставок на странице, что объясняется ниже. Вертикальная плохость вычисляется по тем же правилам, что и горизонтальная: это целое число между 0 и 10000 включительно, но когда бокс переполняется, она равна ∞ (бесконечности).

Когда страница завершена, она из основного вертикального списка посылается в "программу вывода", как мы увидим позже, так что ее боксы и клей исчезают из памяти TEX'а. Остаток основного вертикального списка состоит из двух частей. Первая переходит на "текущую страницу", содержащую весь материал, который TEX рассматривает далее как кандидата для следующей подлежащей разбиению страницы. Затем — "новый вклад", т.е., элементы, которые будут занесены на текущую страницу, когда TEX найдет удобным сделать это. Если вы говорите `\showlists`, TEX выведет содержимое текущей страницы и нового вклада, если они есть, в ваш протокольный файл. (Например, в главе 13 не показано никаких таких списков, потому что они в этом случае были пусты. Глава 24 подробнее объясняет порядок действия TEX'а.)

Всякий раз, когда TEX передвигает элемент из вершины "нового вклада" в нижнюю часть "текущей страницы", он отбрасывает отпадающие элементы (клей, керн, штраф), если текущая страница не содержит ни одного бокса. Вот почему клей исчезает на разрыве страницы. Однако, если отбрасываемый элемент является законной точкой разрыва, TEX вычисляет стоимость разрыва c на этой точке, используя формулы, которые мы только что обсуждали. Если получившееся c меньше или равно наименьшей стоимости, встреченной до сих пор на текущей странице, TEX запоминает текущую точку разбиения как наилучшую на этот момент. А если $c = \infty$ или если $p \le -10000$, TEX перехватывает инициативу и разбивает страницу на наилучшей запомненной точке. Весь материал, следующий на текущей странице за этой наилучшей точкой разрыва, возвращается назад в список нового вклада, где он будет рассматриваться снова. Таким образом, обычно "текущая страница" до того, как выбраны точки разрыва, содержит больше материала, чем требуется для одной страницы.

Эта процедура может показаться таинственной, пока вы не видели ее в действии. К счастью, есть удобный способ понаблюдать за ней: вы можете установить `\tracingpages=1` и этим дать TEX'у инструкции поместить вычисления стоимости страницы в ваш протокольный файл. Например, вот что появилось в протокольном файле, когда автор использовал `\tracingpages=1` в начале этой главы:

```
%% goal height=528.0, max depth=2.2
% t=10.0 g=528.0 b=10000 p=150 c=100000#
% t=22.0 g=528.0 b=10000 p=0 c=100000#
% t=34.0 g=528.0 b=10000 p=100 c=100000#
    ⋮  (25 аналогичных строк здесь опущены)
% t=346.0 plus 1.0 g=528.0 b=10000 p=0 c=100000#
% t=358.0 plus 2.0 g=528.0 b=10000 p=150 c=100000#
% t=370.0 plus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=382.0 plus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=394.02223 plus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=406.02223 plus 2.0 g=528.0 b=10000 p=0 c=100000#
% t=418.0 plus 2.0 g=528.0 b=10000 p=0 c=100000#
% t=430.0 plus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=442.0 plus 2.0 g=528.0 b=10000 p=150 c=100000#
% t=454.02223 plus 2.0 g=528.0 b=10000 p=-100 c=100000#
% t=482.0 plus 5.0 minus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=493.0 plus 5.0 minus 2.0 g=528.0 b=10000 p=0 c=100000#
% t=504.0 plus 5.0 minus 2.0 g=528.0 b=10000 p=100 c=100000#
% t=515.0 plus 5.0 minus 2.0 g=528.0 b=1755 p=150 c=1905#
% t=526.0 plus 5.0 minus 2.0 g=528.0 b=6 p=-100 c=-94#
% t=554.0 plus 8.0 minus 4.0 g=528.0 b=* p=100 c=*
```

Этот результат трассировки имеет не "дружественный" вид, но он из глубины недр TEX'а, где все доведено до числовых вычислений. После небольшой практики можно научиться читать его, но не следует это делать слишком часто, если только вам не надо вникнуть в процесс разбиения страниц для специальных целей. Вот что здесь сказано: первая строка, которая начинается с **%%**, записывается, когда первый бокс или вставка заносятся в текущую страницу списка. Она показывает "нужную высоту" и "максимальную глубину", которые будут использоваться для этой страницы (а именно, текущее значение `\vsize` и `\maxdepth`). В этом руководстве у нас `\vsize=44pc` и `\maxdepth=2.2pt`; размеры в протокольном файле всегда указываются в пунктах (pt). Следующая строка, которая начинается с одиночного **%**, записана, когда разрешенная точка разрыва переносится из списка нового вклада в список текущей страницы. Каждая строка, начинающаяся с **%**, показывает t, которое равно общей высоте до того места, где случился разрыв страницы, и g, которое равно нужной высоте. В этом примере g остается постоянно равным 528 pt, но оно было бы уменьшено, если бы на странице оказалась сноска. Значение t постоянно повышается от 10 к 22, затем к 34 и т.д. Базовые линии отстоят на 12 pt от верха страницы и на 11 pt от низа (если материал задан шрифтом в 9 пунктов). Мы, по существу, видим по одной **%**-строке на h-бокс текста, помещенного на текущей странице. Однако, **%**-строка порождается штрафом и клеем, которые следуют за h-боксами, а не самими боксами. Каждая **%**-строка

показывает также плохость b, штраф p и стоимость c, связанные с точкой разрыва. Если эта стоимость лучше всех, которые были до сих пор, она помечена знаком `#`, означающим, что она будет использована для текущей страницы, если не окажется ничего лучшего. Заметим, что первые 40 или около того разрывов имеют $b = 10000$, так как они настолько плохи, что TEX считает их неразличимыми. В таких случаях $c = 100000$, поэтому TEX просто накапливает материал до тех пор, пока страница не заполнится достаточно, чтобы иметь $b < 10000$. Штраф 150 отражает `\clubpenalty` или `\widowpenalty`, а штраф 100 — штраф за переносы слов, которые вставлены, как описано в главе 14. Две строки с `p=-100` — это точки разрыва между "опасными" абзацами; они берутся из команд `\medbreak`. Записи `b=*` и `c=*` на последней строке означают, что b и c бесконечны. Общая высота 554 pt не может быть преобразована в 528 pt сжатием имеющегося клея. Поэтому страница выведена на наилучшем предыдущем месте, которое привело к вполне хорошему разрыву: `b=6` и `p=-100` дает чистую стоимость -94.

▸ **Упражнение 15.1**
Предположим, что нижний абзац страницы примера на одну строку короче. Какой разрыв был бы выбран?

▸ **Упражнение 15.2**
Две последние `%`-строки показывают, что естественная высота t прыгнула на 28 pt, от 526.0 до 554.0. Объясните, откуда взялся такой большой скачок?

Параметр `\maxdepth` указывает TEX'у поднять нижний бокс страницы, если этот бокс имеет слишком большую глубину, так что глубина сконструированной страницы не будет превышать указанного значения (см. обсуждение `\boxmaxdepth` в главе 12). В нашем примере `\maxdepth=2.2pt`, и влияние этого параметра можно увидеть в строке, которая говорит "`% t=394.02223`". Обычно t в этой точке было бы равно 394.0, но предшествующий ей h-бокс был необычным, потому что содержал букву `j` в `\tt`, а в шрифте в 10 пунктов `j` опускается на 2.22223 pt. Поэтому TEX вычисляет плохость, как если бы h-бокс был на .02223 pt выше, а глубину имел только 2.2 pt.

Заметим, что первая `%`-строка нашего примера говорит `t=10.0`. Это следствие другого параметра, называемого `\topskip`. Клей на разрыве страницы исчезает, но желательно, чтобы базовые линии верхних и нижних строк страниц были, насколько это возможно, в определенном месте, поэтому перед первым боксом каждой страницы TEX вставляет специальный клей. Этот специальный клей равен `\topskip`, если он не уменьшается высотой первого бокса или не устанавливается равным нулю вместо отрицательного значения. Например, если `\topskip=20pt plus2pt` и если первый бокс имеет высоту 13 pt, TEX под этим боксом вставляет `\vskip7pt plus2pt`. Более того, если первый бокс имеет высоту более 20 pt, вставляется`\vskip0pt plus2pt`. Но этот пример не типичен, поскольку клей `\topskip` обычно нерастяжим и несжимаем. Начальный TEX вставляет `\topskip=10pt`.

▸ **Упражнение 15.3**
Предположим, что `\vsize=528pt`, `\maxdepth=2.2pt`, `\topskip=10pt` и что не использованы команды `\insert`. TEX будет получать страницы высотой 528 pt, и обычно справедливы следующие два утверждения: (a) базовая линия верхнего

бокса будет отстоять от верхнего края страницы на 10 pt, т.е., на 518 pt выше базовой линии самой страницы; (b) базовая линия нижнего бокса на странице совпадет с базовой линией самой страницы. Объясните, когда (a) и (b) нарушаются.

Поскольку `\vsize`, `\maxdepth` и `\topskip` являются параметрами, можно в любое время их изменить. Что случится, если вы сделаете это? Ну, TEX использует значения `\vsize` и `\maxdepth`, когда печатает **%%**-строку, т.е., когда на текущей странице появляется первый бокс или вставка. Последующие изменения этих двух параметров не оказывают никакого действия, пока не начнется следующая страница. С другой стороны, TEX смотрит на `\topskip` только тогда, когда на текущую страницу заносится первый бокс. Если вставка оказывается перед первым боксом, то `\topskip`-клей перед этим боксом рассматривается как правильная точка разрыва. Это единственный случай, когда завершенная страница может не содержать бокс.

Можно взглянуть на t и g, которые использовались при разбиении страницы, справившись о значении ⟨размер⟩ в `\pagetotal` и `\pagegoal`, соответственно. Вы можете даже изменить их (но будем надеяться, что вы при этом понимаете, что делаете). Например, команда `\pagegoal=500pt` подавляет предыдущее спасенное значение `\vsize`. Помимо `\pagetotal`, которая равна накопленной естественной высоте, TEX хранит величины `\pagestretch`, `\pagefilstretch`, `\pagefillstretch`, `\pagefilllstretch`, `\pageshrink` и `\pagedepth`. Когда текущая страница не содержит боксов, `\pagetotal` и родственные ей величины равны нулю, а `\pagegoal` равно 16383.99998 pt (наибольший ⟨размер⟩ TEX'a); изменения их значений в таких случаях не действуют. Целое q в формуле для стоимости страницы также доступно для проверки и изменения; оно называется `\insertpenalties`.

Разбиение страниц отличается от разбиения строк еще в одном отношении, которое заслуживает упоминания. Если вы говорите `\eject\eject`, то второй `\eject` игнорируется, потому что это эквивалентно `\penalty-10000`, а штрафы после разбиения страницы сбрасываются. Но если вы скажете в абзаце `\break\break`, то второй `\break` укажет на пустую строку, потому что штрафы сбрасываются после разрыва абзаца, только если они не принадлежат окончательной последовательности точек разрыва. Эта подробность не важна на практике, поскольку `\break\break` — это не самый лучший способ получить пустую строку; такая строка обычно будет недозаполненным h-боксом, поскольку в ней есть только клей, `\leftskip` и `\rightskip`. Аналогично, `\eject\eject` не был бы самым лучшим способом сделать пустую страницу, даже если бы TEX изменил свои правила как-нибудь так, чтобы `\eject` никогда не игнорировался. Наилучший способ выдать пустую страницу — это сказать `\eject\line{}\vfil\eject`, а наилучший способ выдать пустую строку — это `\break\hbox{}\hfil\break`. Оба эти способа избегают недозаполненных боксов.

Вы, вероятно, интересуетесь, как к страницам присоединяются номера страниц и тому подобные штуки. Ответ заключается в том, что TEX позволяет вам производить дальнейшие действия после того, как выбрано разбиение каждой страницы. Перед тем, как страницы в действительности получат свою окончательную форму, вступает в действие специальная "программа вывода". Глава 23 объясняет, как создавать программы вывода и как модифицировать программу вывода начального TEX'a.

Время от времени TEX будет создавать страницы абсолютно ужасного вида, и вы будете удивляться, как это случилось. Например, вы можете получить только один абзац и множество пустого места, в то время, когда некоторый текст, напечатанный на следующей странице, мог бы легко поместиться на этом пустом месте. Причиной такого очевидно ненормального поведения почти всегда служит то, что невозможен хороший разрыв страницы. Даже альтернатива, которая для вас выглядит лучше, кажется абсолютно ужасной, как только в дело вмешивается TEX! TEX не различает две возможности, обе из которых имеют плохость более или равную 10000 единиц, хотя некоторый плохой разрыв выглядит намного лучше других. Решением в таких случаях будет вставить в некотором приемлемом месте `\eject` или `\vfill\eject`, или переработать всю рукопись. Однако если эта проблема возникает часто, то вы, вероятно, используете формат, который накладывает слишком строгие ограничения на форму страницы; попробуйте посмотреть результат команды `\tracingpages` и поменять некоторые параметры TEX'а, пока вам не повезет больше.

Остаток этой главы посвящен вставкам — элементам типа сносок и иллюстраций, и тому, как они взаимодействуют с разрывами страниц. Прежде, чем обсуждать примитивные операции, при помощи которых TEX имеет дело со вставками, мы рассмотрим возможности, которые TEX обеспечивает на более высоком уровне.

Начальный TEX может вставлять иллюстрации несколькими способами. Простейший из них — "плавающая верхняя вставка". Вы говорите

`\topinsert` ⟨вертикальный материал⟩`\endinsert`

и TEX пытается поставить вертикальный материал вверху текущей страницы. ⟨Вертикальный материал⟩ может содержать встроенный абзац, что временно прерывает вертикальную моду обычным способом. Например:

```
\topinsert \vskip 2in
\hsize=3in \raggedright
\noindent{\bf Рисунок 3.} Это надпись над третьей
иллюстрацией моего документа. Я оставил место в два дюйма
под надписью, так что останется место, чтобы вставить специальную
картинку. \endinsert
```

Заголовок в этом примере будет размещен слева на странице в 3-х дюймовой колонке с неровным правым краем. Начальный TEX автоматически добавляет "большой пропуск" ("bigskip") под каждой верхней вставкой; это отделяет заголовок от текста. Действие `\hsize=3in` и `\raggedright` не распространяется после `\endinsert`, поскольку подразумевается группирование.

▸ **Упражнение 15.4**
Измените пример так, чтобы заголовок передвинулся к правому полю, а не печатался слева.

Аналогично, если вы говорите

`\pageinsert`⟨вертикальный материал⟩`\endinsert`

то вертикальный материал будет соответствовать величине полной страницы (без `\bigskip` ниже его); результат появится на следующей странице.

Существует также `\midinsert`⟨вертикальный материал⟩`\endinsert` которая вначале пытается поместить материал в то место, где вы находитесь. Если там есть достаточно места, вы получите следующее:

`\bigskip\vbox{` ⟨вертикальный материал⟩`}\bigbreak`

В противном случае `\midinsert` благополучно преобразуется в `\topinsert`. Есть небольшая вероятность того, что `\midinsert` не найдет наилучшего размещения, потому что TEX иногда обрабатывает текст впереди текущей страницы. Можно сказать `\goodbreak` прямо перед `\midinsert`.

Команды `\topinsert`, `\pageinsert` и `\midinsert` работают в вертикальной моде (т.е., между абзацами), а не внутри боксов или других вставок.

Если две или более команды `\topinsert`, `\pageinsert` или `\midinsert` идут подряд, их можно переносить на несколько последовательных страниц, но при переносе они сохраняют свой относительный порядок. Например, предположим, у вас есть страницы, высота которых равна девять дюймов и вы уже указали 4 дюйма текста на некоторой странице, скажем 25. Затем предположим, вы делаете семь верхних вставок подряд, соответственно, размера $1, 2, 3, 9, 3, 2, 1$ дюйма. 9-дюймовая вставка в действительности представляет собой `\pageinsert`. Что получается? Ну, первая и вторая вставки появятся вверху страницы 25, за ними будет следовать 4 дюйма текста, который вы уже ввели. За этим текстом будет сразу же следовать еще два дюйма текста, который вы ввели после семи вставок. Третья верхняя вставка появится вверху страницы 26, а за ней еще шесть дюймов текста; четвертая заполнит страницу 27, а оставшиеся три появятся вверху страницы 28.

▸ **Упражнение 15.5**
Что случилось бы в только что приведенном примере, если бы последняя вставка была `\midinsert` вместо `\topinsert`?

В конце документа вы, вероятно, хотите, чтобы вставки не исчезли, а в конце главы — чтобы они не залезли в следующую главу. Начальный TEX напечатает все оставшиеся вставки, дополняя пробелами незавершенные страницы. Для этого ему надо сказать `\vfill\supereject`.

Кроме иллюстраций, которые вставляются в верхнюю часть страницы, начальный TEX также умеет вставлять сноски в ее нижнюю часть. Макрокоманда `\footnote` предусмотрена для использования внутри абзацев.* Например, сноска в этом предложении напечатана следующим образом:

```
... абзацев.\footnote*{Как эта.} Например, ...
```

У команды `\footnote` есть два параметра. Первым идет знак ссылки, который появляется как в тексте абзаца**, так и в самой сноске, а затем следует текст сноски.[45] Этот текст может быть длиной в несколько абзацев и содержать выделенные уравнения и тому подобное, но он не должен включать в себя другие

---

* Как эта.

** Автор здесь напечатал “`абзаца\footnote{**}{Автор ...}`”.

[45] Здесь “`сноски \footnote{$^{45}$}{Здесь...}`”. Сноски в этом руководстве печатаются более мелким шрифтом, и для них установлен подвешенный отступ;

вставки. TEX гарантирует, что каждая сноска окажется внизу той страницы, на которой расположена ссылка на нее.† Длинная сноска будет разбита, если это необходимо, и продолжена внизу следующей страницы, как вы могли видеть в до некоторой степени надуманном примере, приведенном здесь. Авторам, которые заинтересованы в хорошем изложении, нужно избегать, насколько возможно, сносок, поскольку сноски действуют отвлекающе.‡

Макрокоманда `\footnote` должна использоваться только в абзацах или h-боксах, которые вносятся в основной вертикальный список TEX'а. Вставки теряются, если они встречаются внутри боксов, находящихся внутри других боксов. Так, например, вы не должны пытаться вставить `\footnote` в подформулу математической формулы. Но если использовать сноски внутри `\centerline`, все будет о'кей. Например,

```
\centerline{Статья A. B. Тора%
  \footnote*{Поддержаная NSF.}}
```

или даже на внешнем уровне элемента таблицы внутри `\halign`.

Верхние вставки прекрасно работают сами по себе, а сноски — сами по себе, но когда вы пытаетесь хитро перемешать их, могут возникнуть осложнения. Например, если `\pageinsert` попадает на страницу, на которой продолжается длинная сноска, обе отложенные вставки могут попытаться втиснуться на одну и ту же страницу и может получиться переполнение v-бокса. Более того, вставки не могут появляться внутри вставок, поэтому нельзя использовать `\footnote` внутри `\topinsert`. Если вам действительно необходима сноска в некоторой подписи, существует макрокоманда `\vfootnote`, которую можно использовать в вертикальной моде. Чтобы использовать ее, поставьте в подписи знак ссылки типа `*`, а затем скажите `\vfootnote*{Сноска}` на той странице, где, как вы предполагаете, эта надпись в конце концов появится. В таких сложных обстоятельствах может быть надо подумать еще раз, действительно ли вы используете самую подходящую форму для демонстрации своих идей.

Глава 24 объясняет точные правила переноса вертикального материала (типа сносок) из горизонтальных списков во внешний вертикальный список. Вставки, метки и результаты `\vadjust` все перемещаются одним и тем же способом.

Теперь давайте изучать примитивы TEX'а, которые использованы для создания таких макрокоманд, как `\topinsert` и `\footnote`. Речь идет о том, чтобы войти за кулисы во внутренний язык TEX'а, который позволяет пользователю совершать сложные манипуляции с боксами и клеем. Наше обсуждение будет

более того, между сносками, печатаемыми на одной странице, делается маленький пропуск. Но в начальном TEX'е сноски печатаются шрифтом нормального размера с `\textindent`, использованным для знака ссылки и без дополнительного пропуска между ними. Макрокоманда `\textindent` похожа на `\item`, но без подвешенного отступа.

† Наборщики часто используют в качестве ссылок символы `\dag` (†), `\ddag` (‡), `\S` (§) и `\P` (¶), а иногда также `$\|$` (‖). Вы можете, например сказать `\footnote\dag{...}`.

‡ Однако книга Гиббона *Наклон и Падение* без сносок стала бы уже не та.

состоять из двух частей. Сначала мы рассмотрим "регистры" TEX'а, с которыми пользователь может производить арифметические действия, имеющие отношение к набору, а затем обсудим вставляемые элементы, которые могут появиться в горизонтальном и вертикальном списках. Наше обсуждение первой части (регистров) будет помечено единичными знаками опасного поворота, поскольку регистры широко используются в сложных приложениях TEX'а, независимо от того, относятся они к вставкам или нет. Вторая же часть будет помечена двойными знаками опасного поворота, поскольку вставки — вещь довольно экзотичная.

TEX имеет 256 регистров, называемых `\count0`, ... , `\count255`, каждый из которых может хранить целое число между -2147483647 и $+2147483647$ включительно, т.е. меньше по абсолютной величине, чем 2^{31}. TEX также имеет 256 регистров, называемых `\dimen0`, ... , `\dimen255`, каждый из которых может содержать ⟨размер⟩ (см.главу 10). Есть другие 256 регистров, называемые `\skip0`, ... , `\skip255`, каждый содержащий ⟨клей⟩ (см. главу 12) и регистры `\muskip0`, ... , `\muskip255`, каждый содержащий ⟨математический клей⟩ (см.главу 18). Вы можете этим регистрам приписать новое значение, сказав

`\count`⟨число⟩ `=` ⟨число⟩
`\dimen`⟨число⟩ `=` ⟨размер⟩
`\skip`⟨число⟩ `=` ⟨клей⟩
`\muskip`⟨число⟩ `=` ⟨математический клей⟩

а также добавить или вычесть величину того же самого типа, сказав

`\advance\count`⟨число⟩ `by` ⟨число⟩
`\advance\dimen`⟨число⟩ `by` ⟨размер⟩
`\advance\skip`⟨число⟩ `by` ⟨клей⟩
`\advance\muskip`⟨число⟩ `by` ⟨математический клей⟩

Например, `\dimen8=\hsize \advance\dimen8 by 1in` устанавливает значение регистра `\dimen8` на 1 дюйм больше, чем текущее значение нормального размера строки.

Если добавляются компоненты бесконечного клея, бесконечности низшего порядка пропадают. Например, после двух команд

```
\skip2 = 0pt plus 2fill minus 3fill
\advance\skip2 by 4pt plus 1fil minus 2filll
```

значением `\skip2` будет 4 pt plus 2 fill minus 2 filll.

Также разрешены умножение и деление, но только нацело. Например, `\multiply\dimen4 by 3` утраивает значение `\dimen4`, а `\divide\skip5 by 2` уменьшает вдвое все три компоненты клея, которые являются текущими значениями регистра `\skip5`. Вы не должны ни делить на ноль, ни умножать на такое число, чтобы результат превышал вместимость регистра. При делении положительного целого на положительное целое остаток отбрасывается, а знак результата изменяется при изменении знака любого операнда. Например, 14 разделенное на 3 дает 4, -14 разделенное на 3 дает -4, -14 разделенное на -3 дает 4. Величины размера являются целыми кратными sp (суперпунктов).

Можно использовать любой регистр `\count` как ⟨число⟩, любой регистр `\dimen` как ⟨размер⟩, любой регистр `\skip` как ⟨клей⟩, а любой `\muskip` как ⟨математический клей⟩. Например, `\hskip\skip1` поместит горизонтальный клей в список, используя значение `\skip1`. А если `\count5` равно 20, то команда `\advance\dimen20 by\dimen\count5` эквивалентна `\multiply\dimen20 by 2`.

Регистр `\dimen` можно также использовать как ⟨число⟩, а `\skip` — как ⟨размер⟩ или ⟨число⟩. TEX преобразует ⟨клей⟩ в ⟨размер⟩, опуская сжимаемость и растяжимость и преобразует ⟨размер⟩ в ⟨число⟩, предполагая единицы sp. Например, если `\skip1` равно 1 pt plus 2 pt, то `\dimen1=\skip1` устанавливает значение `\dimen1` равным 1 pt, а команды `\count2=\dimen1` или `\count2=\skip1` установят `\count2` равным 65536. Эти правила также применимы к внутренним параметрам TEX'а. Например, `\dimen2=\baselineskip` установит `\dimen2` равным естественной компоненте пробела текущего "baselineskip" клея.

▸ **Упражнение 15.6**
Проверьте свои знания регистров TEX'а, сообщив результаты каждой из следующих команд, если они представлены в последовательности:

```
\count1=50   \dimen2=\count1pt   \divide\count1 by 8
\skip2=-10pt plus\count1fil minus\dimen2
\multiply\skip2 by-\count1  \divide\skip2 by \dimen2  \count6=\skip2
\skip1=.5\dimen2 plus\skip2 minus\count\count1fill
\multiply\skip2 by\skip1   \advance\skip1 by-\skip2
```

▸ **Упражнение 15.7**
Что находится в `\skip5` после выполнения следующих трех команд?

```
\skip5=0pt plus 1pt
\advance\skip5 by \skip4  \advance\skip5 by -\skip4
```

▸ **Упражнение 15.8**
(Для математиков.) Объясните, как округлить `\dimen2` до ближайшего кратного `\dimen3`, предполагая, что `\dimen3` не равно нулю.

Регистры подчиняются групповой структуре TEX'а. Например, изменения `\count3` внутри `{...}` не подействуют на `\count3` снаружи. Поэтому TEX в действительности имеет более чем 256 регистров каждого типа. Если вы хотите выполнить регистровскую команду, выходящую за пределы ее группы, то, когда вы изменяете значение, вы должны сказать `\global`.

▸ **Упражнение 15.9**
Что находится в `\count1` после следующей последовательности команд?

```
\count1=5 {\count1=2 \global\advance\count1by\count1
  \advance\count1by\count1}
```

Первые десять регистров `\count`, от `\count0` до `\count9`, зарезервированы для специальных целей: TEX показывает эти десять величин на терминале при выводе страницы и пересылает их в выходной файл как идентификацию этой страницы. Величины отделяются десятичными точками, подавляя лишние ".0". Так, например, если `\count0=5` и `\count2=7`, когда страница отправится

в `dvi`-файл, а другие count-регистры равны нулю, TEX напечатает `[5.0.7]`. Начальный TEX использует `\count0` для номера страницы, а `\count1`.., `\count9` равны нулю. Вот почему, когда страница 1 выведена, вы видите `[1]`. В более сложных приложениях, когда номера страницы могут иметь дополнительную структуру, выводятся десять величин, так что этого хватает для идентификации страницы.

Обычно желательно иметь символические имена для регистров. TEX предусматривает команду `\countdef` (аналогичную `\chardef`, сравните с главой 8), с помощью которой это легко сделать. Вам только надо сказать

```
\countdef\chapno=28
```

и с этого времени `\chapno` будет сокращением для `\count28`. Аналогично, команды `\dimendef`, `\skipdef` и `\muskipdef` годятся для других типов числовых регистров. После того, как команда была определена командой `\countdef`, ее можно использовать в командах TEX'а точно так же, как если бы это был целый параметр типа `\tolerance`. Аналогично, `\dimendef` с успехом создает новый параметр размера, `\skipdef` — новый параметр клея, а `\muskipdef` — новый параметр математического клея.

Кроме числовых регистров у TEX'а также есть 256 боксовых регистров с именами от `\box0` до `\box255`. Боксовые регистры получают значение, когда вы говорите `\setbox`⟨число⟩`=`⟨бокс⟩; например, `\setbox3=\hbox{A}` устанавливает `\box3` в горизонтальный бокс, который содержит одну букву `A`. Несколько других примеров команды `\setbox` уже появлялись в главе 12. В главе 10 отмечалось, что `2\wd3` — это ⟨размер⟩, который равен двойной ширине `\box3`; аналогично, `\ht`⟨число⟩ и `\dp`⟨число⟩ можно использовать, чтобы ссылаться на высоту и глубину данного боксового регистра.

Боксовые регистры являются локальными в группах, так же как и арифметические регистры. Но между боксовыми и всеми остальными регистрами существует большое различие: когда вы используете `\box`, он теряет свое значение. Например, конструкция `\raise2pt\box3` в горизонтальном списке не только помещает содержимое `\box3` в список, подняв его на 2 pt, она также делает `\box3` пустым. TEX делает это для эффективности, поскольку желательно избежать копирования содержимого потенциально больших боксов. Если вы хотите использовать боксовый регистр, не затирая его содержимого, просто скажите `\copy` вместо `\box`; например, `\raise2pt\copy3`.

Другой способ использовать боксовый регистр — сказать `\unhbox`. Это аннулирует содержимое регистра так же, как это делает `\box`, и к тому же устраняет один уровень боксирования. Например, команды

```
\setbox3=\hbox{A} \setbox3=\hbox{\box3 B}
\setbox4=\hbox{A} \setbox4=\hbox{\unhbox4 B}
```

помещают `\hbox{\hbox{A}B}` в `\box3`, а `\hbox{AB}` в `\box4`. Аналогично, `\unvbox` распаковывает v-бокс. Если вы хотите создать большой бокс, наращивая его (например, содержание книги), лучше всего использовать `\unhbox` или `\unvbox`, как в примере `\setbox4`. Иначе вам придется использовать большой объем памяти, а также применять боксы внутри боксов, используя вложенность такой глубины, что будут превышены аппаратные или программные ограничения.

Операции `\unhcopy` и `\unvcopy` относятся к `\unhbox` и `\unvbox`, как `\copy` к `\box`. (Но их имена являются, признаться, странноватыми.)

Операция разбоксирования "размягчает" тот клей, который был установлен на верхнем уровне бокса. Например, рассмотрим последовательность команд.

```
\setbox5=\hbox{A \hbox{B C}} \setbox6=\hbox to 1.05\wd5{\unhcopy5}
```

Она делает `\box6` на пять процентов шире, чем `\box5`. Чтобы сделать это, клей между `A` и `\hbox{B C}` растягивается, но клей во внутреннем h-боксе не меняется.

Боксовый регистр либо "пуст", либо содержит h-бокс или v-бокс. Существует различие между пустым регистром и регистром, который содержит пустой бокс, высота, ширина и глубина которого равны нулю. Например, если `\box3` пустой, вы можете сказать `\unhbox3`, `\unvbox3`, `\unhcopy3` или `\unvcopy3`, но если `\box3` равен `\hbox{}` вы можете сказать только `\unhbox3` или `\unhcopy3`. Если вы говорите `\global\setbox3=`⟨бокс⟩, то когда вы впоследствии используете или разбоксируете `\box3`, он станет "глобально пустым".

▸ **Упражнение 15.10**
Что находится в регистре `\box5` после следующих команд?

```
\setbox5=\hbox{A}  \setbox5=\hbox{\copy5\unhbox5\box5\unhcopy5}
```

▸ **Упражнение 15.11**
Что находится в регистре `\box3` после такой команды:

```
\global\setbox3=\hbox{A}\setbox3=\hbox{}
```

Если вы не уверены в том, как TEX работает со своими регистрами, вы можете проверить это, задавая команды с терминала и используя некоторые команды `\show`. Например,

```
\showthe\count1  \showthe\dimen2  \showthe\skip3
```

покажут содержимое `\count1`, `\dimen2` и `\skip3`, а `\showbox4` покажет содержимое `\box4`. Содержимое бокса появится только в протокольном файле, если только вы не сказали `\tracingonline=1`. Начальный TEX предусматривает макрокоманду `\tracingall`, которая включает все возможные режимы взаимодействия, включая `\tracingonline`. Автор использовал эти команды, чтобы проверить ответы на некоторые из приведенных выше упражнений.

Многие приложения TEX'а используют различные наборы макрокоманд, написанных различными группами людей. Воцарился бы хаос, если бы, скажем, регистр типа `\count100` был одновременно использован в различных макрокомандах для различных целей. Поэтому начальный TEX предусматривает возможность выделения определенного ресурса. Путаница заменится сотрудничеством, если каждый, кто пишет макрокоманду, будет использовать эти соглашения. Идея в том, чтобы сказать, например, `\newcount`, когда вы хотите выделить `\count`-регистр для специальных целей. Например, в этом руководстве автор создал макрокоманду с именем `\exercise` для оформления упражнений, и одна из функций этой команды в том, что она вычисляет номер текущего упражнения.

Макрокоманды формата в приложении E резервируют для этой цели `\count`-регистр, говоря

```
\newcount\exno
```

а затем в начале каждой главы используется команда `\exno=0`. Аналогично, когда появляется новое упражнение, используется команда `\advance\exno by1`, а для печати текущего номера упражнения использована `\the\exno`. Операция `\newcount` назначает уникальный `\count`-регистр своему аргументу `\exno`, а `\exno` определяется командой `\countdef`. Все другие макрокоманды формата написаны не зная, какой `\count`-регистр в действительности соответствует `\exno`.

Кроме `\newcount`, начальный TEX предусматривает `\newdimen`, `\newskip`, `\newmuskip` и `\newbox`; у него так же есть `\newtoks`, `\newread`, `\newwrite`, `\newfam`, и `\newinsert` для целей, которые мы еще не обсуждали. Приложения B и E содержат несколько примеров правильного использования назначений. В случаях `\newbox`, `\newread` и тому подобных назначенный номер определяется при помощи `\chardef`. Например, если команда `\newbox\abstract` используется, чтобы определить боксовый регистр, который будет содержать аннотацию (abstract) и если операция `\newbox` решает для этой цели назначить `\box45`, то она определяет значение `\abstract`, говоря `\chardef\abstract=45`. TEX позволяет использовать величины `\chardef` как целые, так что можно сказать `\box\abstract`, `\copy\abstract` и т.д. (Команды `\boxdef` не существует.)

▸ **Упражнение 15.12**

Напишите макрокоманду `\note`, которая делает последовательно пронумерованные сноски. Например,[1] здесь[2] она должна сделать сноску, если вы вводите

```
... Например,\note{Первое примечание.} здесь\note{Второе примечание.}
она должна сделать сноску, если ...
```

(Используйте `\newcount`, чтобы назначить для сноски регистр `\count`.)

Иногда, однако, вам хочется использовать регистр только для временного заполнения, и вы знаете, что при этом не будет противоречия с какой-нибудь другой командой. Для таких целей традиционно используются регистры `\count255`, `\dimen255`, `\skip255` и `\muskip255`. К тому же начальный TEX резервирует регистры от `\dimen0` до `\dimen9`, от `\skip0` до `\skip9`, от `\muskip0` до `\muskip9` и от `\box0` до `\box9` для "временных пометок"; эти регистры никогда не назначаются операциями `\new...` . Мы видели, что регистры от `\count0` до `\count9` являются специальными, а так же специальным оказывается и `\box255`, поэтому не надо трогать эти регистры, если только вы не знаете точно, что делаете.

Конечно, любой регистр может быть использован для кратковременных целей (включая регистры от `\count0` до `\count9` и `\box255`, а так же включая регистры, предназначенные для других целей), поскольку изменения регистров в группах являются локальными. Однако, вы должны быть уверены, что TEX не будет выдавать какую-нибудь страницу до того, как группа закончится,

---

[1] Первое примечание.

[2] Второе примечание.

поскольку в противном случае программа вывода может быть вызвана в неудачное время. TEX обязан вызывать программу вывода всякий раз, когда пытается переносить что-нибудь из списка нового вклада на текущую страницу, поскольку он мог найти разбиение страницы с $c = \infty$. Приведем список случаев, когда это может произойти. (a) В начале или конце абзаца, при условии, что этот абзац заносится в основной вертикальный список. (b) В начале или конце выделенного уравнения внутри такого абзаца. (c) После завершения `\halign` в вертикальной моде. (d) После занесения бокса, штрафа или вставки в основной вертикальный список. (e) После того, как программа вывода закончилась.

Теперь, вооруженные знаниями о гибких регистрах TEX'а, мы можем погрузиться в детали вставок. Существуют 255 классов вставок от `\insert0` до `\insert254`, и они связаны с регистрами с теми же номерами. Например, `\insert100` связано с `\count100`, `\dimen100`, `\skip100` и `\box100`. Поэтому начальный TEX предусматривает команду назначения для вставок, так же, как он это делает для регистров: приложение B содержит команду

```
\newinsert\footins
```

которая определяет `\footins` как номер для вставки типа сноски. Другие команды, которые имеют дело со сносками, ссылаются на `\count\footins`, `\dimen\footins` и тому подобное. Макрокомандам для плавающих верхних вставок, аналогично, предшествует `\newinsert\topins`, которая определяет `\topins` как номер их класса. Каждый класс вставок независим, но TEX сохраняет порядок вставок внутри класса. Оказывается, что `\footins` имеют класс 254, а `\topins` — класс 253, но макрокоманды не используют эти номера непосредственно.

Для наших целей давайте рассмотрим особый класс вставок, называемый класс n. Мы будем тогда иметь дело с примитивной командой TEX'а

`\insert`n`{` ⟨вертикальный материал⟩`}`,

которая помещает бокс вставки в горизонтальный или вертикальный список. Для этого класса вставок:

`\box`n— здесь появляется материал, когда выведена страница;
`\count`n— коэффициент увеличения для разбиения страницы;
`\dimen`n— размер максимальной вставки на странице;
`\skip`n— дополнительный пробел , назначенный для страницы.

Например, материал, вставляемый с помощью `\insert100`, в конце концов появится в `\box100`.

Пусть естественная высота плюс глубина `\insert`nесть x. Тогда величина `\count`nв 1000 раз больше коэффициента, с которым x действует на размер страницы. Например, начальный TEX задает `\count\footins=1000`, поскольку здесь соотношение один к одному: 10-пунктовая сноска делает страницу на 10 pt короче. Но если у нас приложение, где сноска появляется в две колонки, то для `\count` подходит значение 500. Один из классов вставок в приложении E делает заметки на полях для корректорских целей; в этом случае значение `\count` равно нулю. Никакого реального увеличения не делается; `\count`n— это просто число, используемое для учета, когда оценивается стоимость различных разбиений страниц.


Первая сноска на странице требует дополнительного пробела, поскольку мы хотим отделить сноски от текста и вывести горизонтальную черту. Начальный TEX устанавливает `\skip\footins=\bigskipamount`. Это означает, что программа вывода предполагает добавлять в качестве дополнительного пробела `\bigskip` на любой странице, содержащей, как минимум, одну вставку класса `\footins`.


Иногда желательно наложить максимальные ограничения на размер вставок. Например, обычно не хочется иметь страницы, целиком состоящие из сносок. Начальный TEX устанавливает `\dimen\footins=8in`. Это означает, что `\box\footins` не предполагает накапливать более 8 дюймов вставок для каждой страницы.


Прежде чем читать дальше, вам может захотеться еще раз просмотреть алгоритм разбиения страниц, который объяснялся в начале этой главы. А с другой стороны, может быть, вам вообще не хочется читать остаток этой главы.


Теперь приведем алгоритм, который выполняет TEX, когда `\insert` n переносится из "нового вклада" на "текущую страницу". (Помните, что такой перенос не означает, что вставка в действительности появится на странице; текущая страница будет ограничена позднее точкой разрыва наименьшей стоимости, и только та вставка будет выполнена, которая предшествует этой точке разрыва.) Пусть g и t — текущие значения `\pagegoal` и `\pagetotal`; пусть q — это `\insertpenalties`, накопленный для текущей страницы и пусть d и z — это текущие `\pagedepth` и `\pageshrink`. (Значение d не превышает `\maxdepth`; это значение еще не включено в t.) Наконец, пусть x — естественная высота плюс глубина вставки `\insert`n, которую мы вносим на текущую страницу, и пусть f — соответствующий коэффициент увеличения, т.е. `\count`n, деленное на 1000.

Шаг 1. Если на текущей странице не было предшествующей `\insert`n, g уменьшается на $hf + w$, где h это текущая высота плюс глубина `\box`n, а w — естественная компонента пропуска из `\skip`n; так же включаются компоненты растяжимости и сжимаемости из `\skip`nв итоговые оценки для текущей страницы (в частности, это воздействует на z).

Шаг 2. Если предыдущая `\insert`nна текущей странице была разделена, к q добавляется параметр, называемый `\floatingpenalty`, а шаги 3 и 4 опускаются.

Шаг 3. Проверяется, поместится ли текущая вставка на странице без расщепления. Это означает, что высота плюс глубина `\box`nне превышает `\dimen`n, когда она складывается вместе со всеми предыдущими суммами `\insert`nна текущей странице; более того, это означает, что либо $xf \le 0$, либо $t + d + xf - z \le g$. Если проходят оба теста, xf вычитается из g и опускается шаг 4.

Шаг 4. (Текущая вставка будет расщеплена, по крайней мере для пробы; но расщепление в действительности не имеет места, если страница с наименьшей стоимостью окончится раньше, чем эта вставка.) Сначала вычисляется наибольшая величина v такая, что высота плюс глубина v не сделают в сумме вставки в `\box`nбольшими, чем `\dimen`n, и такими, что $t + d + vf \le g$. (Заметьте, что z в последней формуле опущено, но имеющаяся стоимость рассматривалась в шаге 3, когда мы пытались избежать расщепления.) Затем ищется самый дешевый

способ расщепить начало вертикального списка вставки так, чтобы получить бокс высотой v. (Используется почти такой же алгоритм, как при разбиении страниц, но без стоимости вставки. Предполагается, что в конце вертикального списка присутствует дополнительный элемент `\penalty-10000`, чтобы была уверенность в существовании законной точки разрыва.) Пусть u — естественная высота плюс глубина этого бокса с наименьшей стоимостью и пусть r — штраф, связанный с оптимальной точкой разрыва. Уменьшается g на uf, и увеличивается q на r. (Если `\tracingpages=1`, протокольный файл должен теперь получить загадочное послание, которое говорит `% split` n v`,`u `p=`r. Например,

```
% split254 180.2,175.3 p=100
```

означает, что алгоритм попытался расщепить `\insert254` в 180.2 pt; наилучшее расщепление, оказывается, имеет высоту 175.3 pt, а штраф для разбиения здесь равен 100.)

Этот алгоритм явно усложненный, но, по-видимому, нет более простого механизма, который бы делал хотя бы приблизительно столько же. Заметим, что штраф, равный -10000 внутри вставок, сделает расщепление в шаге 4 очень привлекательным, так что пользователь в трудной ситуации может намекнуть на то, где следует сделать разрыв. Интересно, что этот алгоритм может быть адаптирован для работы множеством различных способов. Плавающие вставки можно сделать специальным случаем расщепляемых вставок, если каждую плавающую вставку начинать с `\penalty0`, так что она свободно может расщепляться и сделать нулевым соответствующий `\floatingpenalty`. Неплавающие вставки типа сносок выделяются установкой большего штрафа за расщепление. (См. приложение B)

Операцию расщепления, упомянутую в шаге 4, можно получить также с примитивной командой: `\vsplit`⟨число⟩ `to`⟨размер⟩ создаст v-бокс, полученный отщеплением указанного количества материала от регистра бокса. Например, команда

```
\setbox200=\vsplit100 to 50pt
```

помещает `\box200` в вертикальный бокс высотой 50 pt. Она просматривает вертикальный список внутри `\box100` (который должен быть вертикальным боксом) и находит разрыв наименьшей стоимости, предполагая целевую высоту в 50 pt, рассматривая плохости и штрафы также, как в случае разбиения страниц (но с $q = 0$). Алгоритм использует `\splitmaxdepth` вместо `\maxdepth`, определяя максимальную глубину боксов. Затем он обрезает верхушку `\box100`, передвигая все наверх и включая все отпадающие элементы, которые следуют непосредственно за оптимальной точкой разрыва. Он использует `\splittopskip`, чтобы вставить новый клей перед первым боксом внутри `\box100`, так же как вверху страницы появляется `\topskip`. Однако если оптимальная точка разрыва окажется в конце вертикального списка внутри `\box100` — предполагается, что там присутствует элемент `\penalty-10000` — или все элементы после оптимальной точки разрыва сброшены, то `\box100` после `\vsplit` будет пустым. А если `\box100` был пустым перед `\vsplit`, то впоследствии будут пустыми оба бокса `\box100` и `\box200`.

Вам лучше не изменять `\box`n, `\count`n, `\dimen`nили `\skip`nв то время, когда TEX заносит вставку на текущую страницу, поскольку алгоритм, описанный выше, предполагает, что эти величины неизменны. Но вы можете поменять `\floatingpenalty`, `\splittopskip` или `\splitmaxdepth`. TEX, когда будет расщеплять и двигать вставки, будет использовать значения, которые были текущими внутри закрывающейся правой скобки в `\insert n{...}`. Например, приложение B в вставках–сносках использует `\floatingpenalty=20000` для того, чтобы охладить пыл сносок, которые расщепляются перед тем, как другие могут начаться; но в плавающих верхних вставках значение `\floatingpenalty` равно нулю. Приложение B также использует специальные величины `\splittopskip` и `\splitmaxdepth` вместе с подпорками (struts), так что расщепленные сноски будут печататься с теми же пробелами, что и нерасщепленные.

Макрокоманда `\footnote` помещает `\insert` в горизонтальный список абзаца. После того, как абзац разбит на строки, эта вставка будет передвинута в вертикальный список сразу после строки, которая ее содержала. (См. главу 14). Поскольку между этим боксом (т.е. этой строкой) и вставкой нет законной точки разрыва, TEX поместит вставку на страницу, которая содержит строку, которая содержит вставку.

▸ **Упражнение 15.13**
Изучите внимательно алгоритм разбиения страниц. Возможно ли, чтобы сноска не появилась на той же странице, где ее ссылка?

Когда наилучшая точка разрыва окончательно выбрана, TEX удаляет из нижней части текущей страницы все, что находится после выбранной точки разрыва и помещает это все назад в верхнюю часть "нового вклада". Сама выбранная точка разрыва помещается на самый верх "нового вклада". Если это штраф, значение штрафа записывается в `\outputpenalty`, а штраф в списке вклада заменяется на 10000; в противном случае `\outputpenalty` устанавливается в нуль. Вставки, которые остались на текущей странице, могут быть трех типов: для каждого класса n — это нерасщепленные вставки, за которыми, возможно, следует простая расщепленная вставка, за которой следуют все остальные. Нерасщепленные вставки добавляются к `\box n` без всякого междустрочного клея. (Должны быть использованы подпорки, как в макрокоманде `\vfootnote` из приложения B.) Если присутствует расщепленная вставка, она `\vsplit` до размера, который предварительно вычислялся на шаге 4; верхняя часть рассматривается как нерасщепленная вставка, а остаток, если имеется, преобразуется во вставку, как-будто он и не отщеплялся. Этот остаток со следующими за ним любыми другими плавающими вставками того же класса откладывается в отдельное место. (Они появятся на "текущей странице", если, когда работает программа вывода, использовано `\showlists`. Общее число таких вставок появляется в `\insertpenalties` во время работы программы вывода. Невставочные элементы перед наилучшим разбиением на текущей странице собираются вместе в `\vbox` высотой g, где g — это `\pagegoal` в момент разрыва, используя запасенное значение `\maxdepth`; этот бокс превращается в `\box255`. Теперь программа вывода пользователя переходит к сканеру TEX'а (см. главу 23). Ее обязанность — собрать окончательные страницы, основанные на содержании `\box255` и всех боксов вставок, о которых ей известно. Программа вывода будет, вероятно, разбоксировать эти боксы так, что

их клей может переустанавливаться. Клей в боксах вставок обычно прекрасно сотрудничает с клеем остальной части страницы, если ему дана такая возможность. После того, как программа вывода закончилась, отложенные вставочные элементы помещаются первыми в список нового вклада, за ними следует вертикальный список, созданный `\output`, а за ними дополнительные вклады, начинающиеся с разбиения страницы. (Глубокий вдох.) Вы усвоили это?

Поскольку нельзя предугадать,
как [сноски] расположатся
в верстке, непрактично нумеровать их
начиная с 1 на каждой странице.
Лучше всего нумеровать их
последовательно во всей статье
или по-главно в книге.

Since it is impossible to foresee
how [footnotes] will happen to come out
in the make-up, it is impracticable
to number them from 1 up on each page.
The best way is to number them
consecutively throughout an article
or by chapters in a book.

— UNIVERSITY OF CHICAGO PRESS, *Manual of Style* (1910)

Не используй в своих книгах сноски, Дон.

Don't use footnotes in your books, Don.

— JILL KNUTH (1962)

16

Печать математики

TEX разработан, чтобы обращаться со сложными математическими выражениями таким образом, чтобы большинство из них было легко вводить. Основная идея в том, что сложные формулы собираются из менее сложных формул, составляя их простым способом. Менее сложные формулы, в свою очередь, сделаны из простых комбинаций формул еще меньшей сложности, и т.д. Можно сказать иначе: если вы знаете, как вводить простые формулы и как их комбинировать в более крупные, то вы сможете обращаться фактически с любыми формулами. Поэтому давайте начинать с простого и постепенно разрабатывать наш способ.

Простейшая формула — это буква типа x или число типа 2. Для того, чтобы вставить их в текст TEX'а, вы, соответственно, вводите `$x$` и `$2$`. Заметим, что все математические формулы заключены в специальные математические скобки. Согласно формату начального TEX'а, как он определен в приложении B, в качестве математических скобок используется знак доллара `$`, ибо математика должна стоить дорого.

Когда вы вводите `$x$`, то x печатается курсивом, но когда вы вводите `$2$`, то 2 печатается прямым шрифтом. Вообще, все символы вашей клавиатуры имеют в математических формулах специальную интерпретацию в соответствии с обычными соглашениями математических изданий. Буквы здесь обозначают курсивные буквы, в то время как цифры и знаки пунктуации обозначают латинские цифры и пунктуацию. Знак (`-`) теперь означает знак минус ($-$), который похож на em-тире, но не совпадает с ним (см. главу 2). Первый знак `$`, который вы печатаете, переводит в "математическую моду, а второй — возвращает обратно. (см. главу 13). Так что, если вы забыли один `$` или напечатали слишком много `$`, TEX, вероятно, совершенно запутается, и вы получите некоторое сообщение об ошибке.

Формула, которую напечатал наборщик, незнакомый с математикой, для математика выглядят очень странно, поскольку начинающий наборщик обычно все пробелы ставит неправильно. Для того, чтобы сгладить эту проблему, TEX большинство пробелов в формулах делает сам и *игнорирует* любые пробелы, которые вы сами поставили между знаками `$`. Например, если вы вводите `$ x$` и `$ 2 $`, то это будет означать то же самое, что `$x$` и `$2$`. Можно ввести `$(x + y)/(x - y)$` или `$(x+y)/(x-y)$`, но в обоих случаях в результате будет $(x+y)/(x-y)$, т.е., формула, в которой знаки $+$ и $-$ окружены небольшими дополнительными пробелами, а знак `/` — нет. Таким образом, вы не должны запоминать сложные правила распределения математических пробелов и можете использовать пробелы любым способом, который вам нравится. Конечно, пробелы используются еще и для обычных целей, чтобы отметить конец команд, как говорилось в главе 3. В большинстве случаев пробелы TEX'а будут привычными для математика, но в главе 18 мы увидим, что существуют команды, которыми вы, если захотите, сможете подавить правила TEX'а для расстановки пробелов.

Что еще математики любят — так это делать свои формулы похожими на древнегреческие. На языке начального TEX'а вы можете ввести

`$$\alpha,\beta, \gamma, \delta;$$` и получите первые четыре греческие буквы

$$\alpha, \beta, \gamma, \delta.$$

Более того, есть заглавные греческие буквы, такие как Γ, которые вы можете получить, вводя `$\Gamma$`. Не пугайтесь, если вы еще не знакомы с греческими буквами; если они вам понадобятся, их легко выучить. Единственная трудность в том, что некоторые символы, которые выглядят почти одинаково, нужно внимательно различать. Например, греческие буквы `\nu` (ν) и `\kappa` (κ) нельзя путать с курсивными буквами v и x. Греческая `\phi` (ϕ) отличается от наклонного нуля `\emptyset` ($\emptyset$). Строчная эпсилон (ϵ) совершенно отличается от символа, который обозначает включение в множество ($\in$); надо вводить `$\epsilon$` чтобы получить ϵ и `$\in$` для $\in$. Некоторые строчные греческие буквы имеют в математических курсивных шрифтах начального TEX'а различные формы: `$(\phi,\theta,\epsilon,\rho)$` дает $(\phi, \theta, \epsilon, \rho)$, в то время как `$(\varphi,\vartheta,\varepsilon,\varrho)$` дает $(\varphi, \vartheta, \varepsilon, \varrho)$.

Кроме греческих букв, существует много интересных символов типа $\approx$ (который можно получить, вводя `$\approx$`) и $\mapsto$ (который можно получить, вводя `$\mapsto$`). Полный список таких команд и соответствующих им символов приведен в приложении F. Такие команды разрешены только в математической моде, т.е. между знаками $, потому что соответствующие им символы относятся к математическим шрифтам.

▸ **Упражнение 16.1**
Что вы должны ввести, чтобы получить формулу $\gamma + \nu \in \Gamma$?

▸ **Упражнение 16.2**
Посмотрите приложение F, чтобы узнать команды для $\leq$, $\geq$ и $\neq$. (Это, вероятно, наиболее часто используемые математические символы, которых нет у вас на клавиатуре.) Как начальный TEX вызывает их?

Теперь давайте посмотрим, как из простых формул строятся более сложные. Во-первых, вы можете получить верхний индекс $^{\text{(вверху(high))}}$ и нижний индекс $_{\text{(внизу(low))}}$ используя `^` и `_`, как показано в следующих примерах:

Вход	Выход
`$x^2$`	x^2
`$x_2$`	x_2
`$2^x$`	2^x
`$x^2y^2$`	x^2y^2
`$x ^ 2y ^ 2$`	x^2y^2
`$x_2y_2$`	x_2y_2
`$_2F_3$`	${}_2F_3$

Заметим, что `^` и `_` применяются только к одному следующему за ними символу. Если вы хотите в качестве верхнего или нижнего индекса использовать несколько символов, заключите их в фигурные скобки:

`$x^{2y}$`	x^{2y}
`$2^{2^x}$`	2^{2^x}
`$2^{2^{2^x}}$`	$2^{2^{2^x}}$
`$y_{x_2}$`	y_{x_2}
`$y_{x^2}$`	y_{x^2}

Фигурные скобки здесь используются, чтобы указать "подформулы", т.е. простые части более сложной формулы. TEX для каждой подформулы создает бокс и рассматривает этот бокс, как если бы он был одним символом. Фигурные скобки служат и для обычных целей группирования, как обсуждалось в главе 5.

Нельзя вводить `x^y^z` или `x_y_z`; TEX будет обижаться на "двойной верхний индекс" или "двойной нижний индекс". Для того, чтобы четко выразить ваши намерения, вы должны ввести `x^{y^z}`, `x^{yz}`, `x_{y_z}` или `x_{yz}`.

Верхний или нижний индекс, за которым следует символ, применяется только к этому символу, но когда за индексом следует подформула, он применяется к целой подформуле, и эта подформула будет, соответственно, поднята или опущена. Например,

`$((x^2)^3)^4$`	$((x^2)^3)^4$
`${{(x^2)}^3)}^4$`	$\left((x^2)^3\right)^4$

В первом примере `^3` и `^4` — это верхние индексы над правыми скобками, т.е. над символами `)`, которые находятся перед ними, а во второй формуле — это индексы над подформулами, которые заключены в фигурные скобки. Первая альтернатива предпочтительней, потому что ее легче как вводить, так и читать.

Верхние или нижние индексы, перед которыми нет ничего (как в примере `_2F_3` на предыдущей странице, где перед `_2` ничего нет) считаются верхними или нижними индексами пустой подформулы. Такая запись (к счастью) в математике встречается редко, но если вы сталкиваетесь с ней, лучше ясно выразить свое пожелание, явно показав фигурными скобками пустую подформулу. Другими словами, чтобы в формуле получить ${}_2F_3$, лучше всего ввести `{}_2F_3`, `{_2}F_3` или `{_2F_3}`.

▸ **Упражнение 16.3**

Какое различие, если оно есть, между результатом `$x + _2F_3$` и `$x + {}_2F_3$`?

▸ **Упражнение 16.4**

Опишите различия между результатами `${x^y}^z$` и `$x^{y^z}$`.

Можно иметь одновременно верхний и нижний индексы и указывать их в любом порядке:

`$x^2_3$`	x^2_3
`$x_3^2$`	x^2_3
`$x^{31415}_{92}+\pi$`	$x^{31415}_{92}+\pi$
`$x_{y^a_b}^{z_c^d}$`	$x^{z_c^d}_{y^a_b}$

Заметим, что одновременные индексы$^{\text{верхний}}_{\text{нижний}}$ располагаются один под другим. Однако нижний индекс, когда он следует за некоторыми буквами, будет слегка "втянутым": например, `$P_2^2$` производит P_2^2. Если по каким-то причинам вы хотите, чтобы левые края верхнего и нижнего индекса были выровнены, вы можете обмануть TEX, вставив нулевую подформулу: `$P{}_2^2$` дает $P{}_2^2$.

Команда `\prime` обозначает символ `'`, который в основном используется в качестве верхнего индекса. Действительно, `'` сам по себе настолько велик, что вы не захотите использовать его за исключением верхних и нижних индексов, где он встречается в уменьшенном размере. Приведем несколько типичных примеров:

Вход	Выход
`$y_1^\prime$`	$y_1^\prime$
`$y_2^{\prime\prime}$`	$y_2^{\prime\prime}$
`$y_3^{\prime\prime\prime}$`	$y_3^{\prime\prime\prime}$

Поскольку простые и двойные штрихи встречаются довольно часто, начальный TEX предусматривает удобное сокращение: можно просто вводить `'` вместо `^\prime`, `''` вместо `^{\prime\prime}`, и так далее.

`$f'[g(x)]g'(x)$`	$f'[g(x)]g'(x)$
`$y_1'+y_2''$`	$y_1'+y_2''$
`$y'_1+y''_2$`	$y'_1+y''_2$
`$y'''_3+g'^2$`	$y'''_3+g'^2$

▸ **Упражнение 16.5**

Почему, как вы думаете, TEX трактует `\prime` как большой символ, который появляется только в верхних индексах, вместо того, чтобы сделать его маленьким и уже помещенным в положение верхнего индекса?

▸ **Упражнение 16.6**

Математики иногда используют "тензорную запись",в которой верхние и нижние индексы расположены этапами, как в ${R_i}^{jk}{}_l$. Объясните, как получить такой эффект.

Другой способ получения из простых формул сложной — это использовать команды `\sqrt`, `\underline` или `\overline`. Так же, как `^` и

_, эти операции применяются к символу или подформуле, которые за ними следуют:

`$\sqrt2$`	$\sqrt2$
`$\sqrt{x+2}$`	$\sqrt{x+2}$
`$\underline4$`	$\underline{4}$
`$\overline{x+y}$`	$\overline{x+y}$
`$\overline x+\overline y$`	$\overline{x}+\overline{y}$
`$x^{\underline n}$`	$x^{\underline{n}}$
`$x^{\overline{m+n}}$`	$x^{\overline{m+n}}$
`$\sqrt{x^3+\sqrt\alpha}$`	$\sqrt{x^3+\sqrt{\alpha}}$

Можно также получать кубический корень $\sqrt[3]{\ }$ и аналогичные вещи, используя `\root`:

`$\root 3 \of 2$`	$\root 3 \of 2$
`$\root n \of {x^n+y^n}$`	$\root n \of {x^n+y^n}$
`$\root n+1 \of a$`	$\root n+1 \of a$

Операции `\sqrt`, `\underline` и `\overline` могут поместить черту над или под подформулами любого размера или формы; черта меняет свой размер и положение, так что она достаточно длинна, чтобы покрыть подформулу и расположена на достаточной высоте, чтобы не наехать на нее. Например, рассмотрим `\overline l` ($\overline l$) по сравнению с `\overline m` ($\overline m$). В первом случае черта короче и выше поднята, чем во втором. Аналогично, черта в `\underline y` ($\underline y$) ниже, чем черта в `\underline x` ($\underline x$); знаки квадратного корня появляются в различных видах, в зависимости от высоты, глубины и ширины выражения, из которого извлекается квадратный корень: $\sqrt a+\sqrt d+\sqrt y$. TEX знает высоту, глубину и ширину каждой буквы и каждой подформулы, потому что рассматривает их как боксы, как это объяснялось в главе 11. Если у вас есть формула, в которой только один `\sqrt` или только один `\overline` или `\underline`, нормальные правила позиционирования работают прекрасно, но иногда надо иметь единообразие между различными частями сложной формулы. Например, можно так ввести $\sqrt a+\sqrt d+\sqrt y$, чтобы все квадратные корни были одинаковой высоты. Легко сделать это, используя команду `\mathstrut`, как показано далее:

```
$\sqrt{\mathstrut a}+\sqrt{\mathstrut d}+\sqrt{\mathstrut y}$.
```

`\mathstrut` — это невидимый бокс, ширина которого равна нулю, а высота и глубина равны высоте и глубине скобки (. Поэтому подформулы, которые содержат `\mathstrut`, будут всегда иметь одинаковую высоту и глубину, если только они не включают в себя более сложные конструкции типа верхних и нижних индексов. Глава 18 обсуждает более мощные операции, называемые `\smash` и `\phantom`, с помощью которых можно добиться полного контроля над положением корней и аналогичных знаков.

▸ **Упражнение 16.7**

Проверьте, как вы поняли то, что прочли в этой главе, объяснив, что долж-

но быть введено, чтобы получить следующие формулы. (Чтобы быть уверенным сверьте ваши ответы с приложением A для подтверждения своей правоты.)

$$10^{10} \quad 2^{n+1} \quad (n+1)^2 \quad \sqrt{1-x^2} \quad \overline{w+\overline{z}} \quad p_1^{e_1} \quad a_{b_{c_{d_e}}} \quad \sqrt[3]{h_n''(\alpha x)}$$

▸ **Упражнение 16.8**
Какую ошибку обнаружил B.C. Dull после того, как ввел следующее:

```
Если$ x = y$, то $x$ равно $y.$
```

▸ **Упражнение 16.9**
Объясните, как ввести следующее предложение:

Удаляя элемент из n-множества получаем $(n-1)$-множество.

▸ **Упражнение 16.10**
Перечислите все буквы курсива, которые опускаются ниже нижней линии шрифта. (Это буквы, для которых черта, полученная `\underline` будет ниже.)

Мы обсудили тот факт, что символы, которые вы вводите, в математической моде имеют специальные значения, но примеров этого до сих пор приведено недостаточно; еще не раскрыта вся власть, которая заключается в кончиках ваших пальцев после того, как вы нажали клавишу `$`. Сейчас настало время вернуться к основам: давайте проведем систематический обзор того, что делает каждый символ, когда он используется в формуле.

52 буквы (от `A` до `Z` и от `a` до `z`) обозначают символы курсива (от A до Z и от a до z), которые математики называют "переменными". TEX называет их просто "обычными символами", потому что они составляют большую часть математических формул. В начальном TEX'е существует два варианта буквы L нижнего регистра, а именно, l и ℓ (которую вы получаете, просто вводя `\ell`). Математики в своих рукописях обычно пишут что-то похожее на ℓ, но делают это единственно для того, чтобы отличить ее от цифры 1. Этой проблемы нет в печатных математических работах, поскольку курсивная l сильно отличается от 1, поэтому принято использовать l вместо того, чтобы специально запрашивать ℓ.

Начальный TEX также трактует 18 символов

```
0 1 2 3 4 5 6 7 8 9 ! ? . | / ‘ @ "
```

как обычные символы, т.е. не вставляет никаких дополнительных пробелов, когда эти символы следуют один за другим или рядом с буквами. В отличие от букв, эти 18 символов, когда появляются в формулах, остаются в прямом шрифте. Вам не надо о них помнить ничего особенного, за исключением того, что вертикальная черта | используется специальным образом, что мы обсудим позднее. Кроме того, вам надо быть внимательным, чтобы отличать O и ноль. Буква курсива O почти никогда не используется

в формулах, кроме случаев, когда она появляется непосредственно перед левыми скобками, как в $O(n)$; а цифра 0 почти никогда не используется перед левыми скобками, если только ей не предшествует другая цифра, как в $10(n-1)$. Следите за левой скобкой и все будет о'кей. (Строчная о также в основном появляется только перед левыми скобками: вводите `x_0` вместо `x_o`, поскольку формула x_0 обычно более правильная, чем x_o.)

Три символа `+`, `-`, и `*` называются "бинарными операциями", потому что они оперируют с двумя частями формулы. Например, `+` — это знак плюс, который используется для суммы двух чисел; `-` — это знак минус. Звездочка (`*`) в математике используется редко, но она тоже ведет себя, как бинарная операция. Приведем несколько примеров того, как TEX печатает бинарные операции, когда они появляются рядом с обычными символами:

Вход	Выход
`$x+y-z$`	$x+y-z$
`$x+y*z$`	$x+y*z$
`$x*y/z$`	$x*y/z$

Заметим, что `-` и `*` дают математические символы, абсолютно отличные от тех, которые вы получаете в обычном тексте. Знак дефис (-) становится знаком минуса ($-$), а поднятая звездочка (*) опускается на более низкий уровень ($*$).

TEX не рассматривает знак `/`, как бинарную операцию, хотя он обозначает деление (которое в математике считается бинарной операцией). Причина в том, что наборщики традиционно ставят дополнительные пробелы вокруг символов $+$, $-$ и $*$ и не ставят их вокруг $/$. Если бы TEX печатал `/`, как бинарную операцию, то формула `$1/2$` получилась бы в виде $1/2$, что было бы неправильно; поэтому TEX рассматривает `/` как обычный символ.

Приложение F приводит много других бинарных операций, для которых вы печатаете команды, а не единичные символы. Приведем несколько примеров:

`$x\times y\cdot z$`	$x\times y\cdot z$
`$x\circ y\bullet z$`	$x\circ y\bullet z$
`$x\cup y\cap z$`	$x\cup y\cap z$
`$x\sqcup y\sqcap z$`	$x\sqcup y\sqcap z$
`$x\vee y\wedge z$`	$x\vee y\wedge z$
`$x\pm y\mp z$`	$x\pm y\mp z$

Важно отличать $\times$ (`\times`) от X (`X`) и от x (`x`), отличать $\cup$ (`\cup`) от U (`U`) и от u (`u`), отличать $\vee$ (`\vee`) от V (`V`) и от v (`v`), отличать $\circ$ (`\circ`) от O (`O`) и от o (`o`). Символы $\vee$ и $\wedge$ могут быть также вызваны `\lor` и `\land`, поскольку они часто обозначают бинарные операции, называемые "логическое или" и "логическое и".

Случается, что бинарные операции трактуются как обычные символы, если они встречаются не между двумя величинами, с которыми они опери-

руют. Например, в таких случаях, как приведенные ниже, не вставляются никакие дополнительные пробелы рядом с $+$, $-$ и $*$:

`$x=+1$`	$x = +1$
`$3.142-$`	$3.142-$
`$(D*)$`	$(D*)$

Рассмотрим также следующие примеры, которые показывают, что бинарные операции могут быть использованы как обычные символы в верхних и нижних индексах:

`$K_n^+,K_n^-$`	K_n^+, K_n^-
`$z^*_{ij}$`	z^*_{ij}
`$g^\circ \mapsto g^\bullet$`	$g^\circ \mapsto g^\bullet$
`$f^*(x) \cap f_*(y)$`	$f^*(x) \cap f_*(y)$

▸ **Упражнение 16.11**
Как бы вы могли получить формулы z^{*2} и $h'_*(z)$?

Начальный TEX трактует символы `=`, `<`, `>` и `:` как "отношения", потому что они выражают отношение между двумя величинами. Например, $x < y$ означает, что x меньше, чем y. Такие отношения значительно отличаются по смыслу от бинарных операций типа $+$, и эти символы печатаются несколько иначе:

`$x=y>z$`	$x = y > z$
`$x:=y$`	$x := y$
`$x\le y\ne z$`	$x \le y \ne z$
`$x\sim y\simeq z$`	$x \sim y \simeq z$
`$x\equiv y\not\equiv z$`	$x \equiv y \not\equiv z$
`$x\subset y\subseteq z$`	$x \subset y \subseteq z$

(Последние несколько примеров показывают некоторые из многих других символов отношения, которые начальный TEX делает доступными через команды, см. приложение F).

Два символа "`,`" (запятая) и "`;`" (точка с запятой) трактуются в формулах как знаки пунктуации; это означает, что TEX ставит небольшой дополнительный пробел после них и не ставит до них.

`$f(x,y;z)$`	$f(x,y;z)$

В математических формулах не принято ставить дополнительный пробел после "`.`" (точки), поэтому TEX трактует точку как обычный символ. Если вы хотите, чтобы символ "`:`" трактовался как знак пунктуации, а не как отношение, просто вызывайте его командой `\colon`:

`$f:A\to B$`	$f : A \to B$
`$f\colon A\to B$`	$f\colon A \to B$

Если вы хотите использовать запятую как обычный символ (например, когда она появляется в большом числе), поставьте ее в фигурных скобках; TEX трактует все, что находится в фигурных скобках как обычный символ. Например,

`$12,345x$`	$12,345x$	(неправильно)
`$12{,}345x$`	$12{,}345x$	(правильно)

▸ **Упражнение 16.12**
Как легко получить поднятую точку в десятичной константе (например, 3·1416)?

Мы рассмотрели буквы, другие обычные символы, бинарные операции, отношения и другие знаки пунктуации, следовательно, охватили почти все клавиши пишущей машинки. Их немногим больше: символы `(` и `[` называются "открывающими", а символы `)`, `]` — "закрывающими"; они прекрасно действуют как обычные символы, к тому же помогают TEX'у решить, когда бинарные операции в действительности не используются бинарным способом. Затем существует символ `'`, который, как мы знаем, используется как сокращение для верхнего индекса `\prime`. Наконец, мы знаем, что начальный TEX резервирует еще десять символов:

```
\ $ % # & ~ { } _ ^
```

Они не используются для символов в математической моде, если только не было изменено значение их `\catcode` (см. главу 7). Хотя `{` и `}` указывают группирование, команды `\{` и `\}` можно использовать, чтобы получить { открывающую и } закрывающую.

Все эти интерпретации математической моды легко изменяемы, поскольку, как объясняется в главе 17, каждый символ имеет `\mathcode`. Ни одно соглашение не встроено в TEX неизменным. Однако большинство из них настолько стандартны, что обычно неразумно делать много изменений, за исключением, может быть, интерпретации `‘`, `"`, `@`.

Специальные символы `^` и `_`, которые обозначают верхние и нижние индексы, не должны использоваться вне формул. Аналогично, имена математических символов типа `\alpha` и `\approx` и команды для математических операций типа `\overline` не должны вторгаться в обычный текст. TEX использует такие факты, чтобы обнаружить в вашем входном файле пропущенный знак доллара перед тем, как такие ошибки вызовут слишком много неприятностей. Например, предположим, что вы ввели

```
Наименьшее $n такое, что $2^n>1000$ равно~10.
```

TEX не знает, что вы забыли `$` после первой `n`, потому что не понимает русского языка; поэтому он считает "формулой" то, что находится между первыми двумя знаками `$`:

Наименьшее nтакое, что

после чего он думает, что 2 — это часть текста. Но затем несообразно обнаруживается `^`; TEX автоматически вставит `$` перед `^`, и вы получите сообщение об ошибке. В этом случае компьютер переходит в режим ожидания, и может быть напечатана остальная часть документа, если не случилось ничего другого.

Наоборот, пустая строка или `\par` в математической моде не разрешены. Это предоставляет TEX'у другой способ оправиться от пропуска `$`; такие ошибки будут содержаться в пределах абзаца, в котором они встретились.

Если по некоторым причинам вы не можете использовать `^` и `_` для верхних и нижних индексов, потому что пользуетесь необычной клавиатурой или потому что вам нужен `^` для французского акцента или для чего-нибудь еще, начальный TEX позволяет вместо них вводить `\sp` и `\sb`. Например, `$x\sp2$` — это еще один способ получить x^2. С другой стороны, некоторым посчастливилось иметь клавиатуры, которые кроме стандартных символов ASCII содержат дополнительные символы. Когда доступны такие символы, TEX может сделать математическую печать намного более приятной. Например, в устройстве автора существуют клавиши, помеченные ↑ и ↓ , которые производят видимые символы (они делают так, что верхние и нижние индексы выглядят на экране намного лучше); существуют клавиши для отношений ≤, ≥ и ≠ (они экономят время) и еще около двух дюжин клавиш, которые могут пригодиться (см. приложение C).

Математики любят использовать акценты над буквами, потому что это часто бывает эффективным способом указать связь между математическими объектами и сильно расширяет количество доступных символов без увеличения количества необходимых шрифтов. Глава 9 обсуждает, как использовать акценты в обычном тексте, но математические акценты — это особый случай, потому что здесь другая расстановка пробелов: TEX для акцентов в формулах использует специальные соглашения, так что два вида акцентов не следует путать друг с другом. Начальным TEX'ом предусмотрены следующие математические акценты:

`$\hat a$`	$\hat{a}$
`$\check a$`	$\check{a}$
`$\tilde a$`	$\tilde{a}$
`$\acute a$`	$\acute{a}$
`$\grave a$`	$\grave{a}$
`$\dot a$`	$\dot{a}$
`$\ddot a$`	$\ddot{a}$
`$\breve a$`	$\breve{a}$
`$\bar a$`	$\bar{a}$
`$\vec a$`	$\vec{a}$

Первые девять из них, когда они появляются в тексте, вызываются, соответственно, `\^`, `\v`, `\~`, `\'`, `\‘`, `\.`, `\"`, `\u`, и `\=`; `\vec` — это акцент, который появляется только в формулах. TEX возражает, если вы пытаетесь в формулах использовать `\^` или `\v` и так далее, или в обычном тексте использовать `\hat` или `\check` и тому подобное.

Обычно удобно определить специальные команды для акцентированных букв, которые вам часто нужны. Например, можно установить

```
\def\Ahat{{\hat A}}
\def\chat{{\hat c}}
\def\scheck{{\check s}}
\def\xtilde{{\tilde x}}
\def\zbar{{\bar z}}
```

в начале рукописи, которая использует символы $\hat{A}$, $\hat{c}$, $\check{s}$, $\tilde{x}$ и $\bar{z}$ более чем, скажем, пять раз. Это избавит вас от множества нажатий клавиш и сделает рукопись более легкой для чтения. Глава 20 объясняет, как определять команды.

Когда в математических формулах акцентируются буквы i и j, то под акцентами должны должны использоваться бесточечные символы $\imath$ и $\jmath$. Эти символы вызываются в начальном TEX'е при помощи `\imath` и `\jmath`. Так, например, документ, который использует $\hat{\imath}$ и $\hat{\jmath}$, должен начинаться со следующих определений:

```
\def\ihat{{\hat\imath}}
\def\jhat{{\hat\jmath}}
```

Вы можете поставить акцент над акцентом и создать такой символ, как $\hat{\hat{A}}$, который заставит математика визжать в экстазе. Однако, чтобы получить верхний акцент правильно, потребуются некоторые ухищрения, т.к. разработчик шрифта для математики обычно указывает TEX'у положение математических акцентов специальным способом для специальных букв. Начальный TEX предусматривает команду `\skew`, которая легко подвигает верхний акцент на подходящее ему место. Например, чтобы получить приведенный выше символ, использована команда `\skew6\hat\Ahat`. Число `6` в этом примере было выбрано путем проб и ошибок; `5` означает поставить верхний акцент немного левее, а `7` делало его слишком правым, по крайней мере, по мнению автора. Идея в том, чтобы поиграть с величиной отклонения, пока вы не найдете то, что вам понравится больше всего.

Фактически, можно поставить математический акцент над любой подформулой, а не только над простыми и акцентированными символами. Но обычно не стоит так делать, потому что TEX центрирует акцент над целой формулой. Например, `$\hat{I+M}$` дает $\hat{I+M}$. В частности, акцент `\bar` всегда остается одного и того же размера, не так как `\overline`, которая растет над формулой вместе с ней. Некоторые предпочитают более длинную черту, полученную `\overline`, даже когда она применяется к одиночной букве. Например, `$\bar z+\overline z$` производит $\bar{z} + \overline{z}$, и вы можете сделать свой выбор, определив `\zbar`. Однако начальный TEX предусматривает два акцента, которые растут; они вызываются `\widehat` и `\widetilde`:

`$\widehat x,\widetilde x$`	$\widehat{x}, \widetilde{x}$
`$\widehat{xy},\widetilde{xy}$`	$\widehat{xy}, \widetilde{xy}$
`$\widehat{xyz},\widetilde{xyz}$`	$\widehat{xyz}, \widetilde{xyz}$

Третий пример здесь показывает максимальный возможный размер.

▸ **Упражнение 16.13**

Это была еще одна длинная глава. Но утешьтесь, вы много узнали! Докажите это, объяснив, что надо ввести, чтобы получить формулы e^{-x^2}, $D \sim p^{\alpha}M + l$, и $\hat{g} \in (H^{\pi_1^{-1}})'$. (В последнем примере предполагается, что определена команда `\ghat` так, что `\ghat` производит акцентированную букву $\hat{g}$.)

Делать греческие буквы так же легко,
как π*. Печатайте просто*
`... легко, как $\pi \pi$.`

Producing Greek letters
is as easy as π*. You just type*
`... as easy as $\pi$.`

—LESLIE LAMPORT, *The LaTeX Document Preparation System* (1983)

TEX'у безразлична слава греческого языка, для него греческие буквы — это всего лишь дополнительные причудливые символы, доступные только в математической моде. Если нужно, вы можете выдать слово $\tau\epsilon\chi$*, печатая* `$\tau\epsilon\chi$`*, но если бы вы действительно набирали греческий текст, то использовали бы другую версию TEX'а, рассчитанную на клавиатуру с греческими буквами, и даже вряд ли стали бы читать это руководство, которое для вас является насквозь английским.*

TEX has no regard for the glories of the Greek tongue—as far as it is concerned, Greek letters are just additional weird symbols, and they are allowed only *in math mode. In a pinch you can get the output* $\tau\epsilon\chi$ *by typing* `$\tau\epsilon\chi$`*, but if you're actually setting Greek text, you will be using a different version of TEX, designed for a keyboard with Greek letters on it, and you shouldn't even be reading this manual, which is undoubtedly all English to you.*

— MICHAEL SPIVAK, *The Joy of TEX* (1982)

17

Еще о математике

А еще математики любят дроби, а также символы, написанные один над другим множеством различных способов:

$$\frac{1}{2} \quad \text{и} \quad \frac{n+1}{3} \quad \text{и} \quad \binom{n+1}{3} \quad \text{и} \quad \sum_{n=1}^{3} Z_n^2 .$$

Можно получить эти формулы, если ввести `$$1\over2$$`, `$$n+1\over3$$`, `$$n+1\choose3$$` и `$$\sum_{n=1}^3 Z_n^2$$`. В этой главе мы будем изучать простые правила для создания таких конструкций.

Сначала рассмотрим дроби, которые используют запись `\over`. Команда `\over` применяется ко всей формуле, если только вы не используете фигурные скобки, чтобы заключить что-то в специфическую подформулу. В последнем случае `\over` применяется ко всей подформуле.

Вход	Выход
`$$x+y^2\over k+1$$`	$$x+y^2\over k+1$$
`$${x+y^2\over k}+1$$`	$${x+y^2\over k}+1$$
`$$x+{y^2\over k}+1$$`	$$x+{y^2\over k}+1$$
`$$x+{y^2\over k+1}$$`	$$x+{y^2\over k+1}$$
`$$x+y^{2\over k+1}$$`	$$x+y^{2\over k+1}$$

Нельзя использовать `\over` в одной и той же подформуле дважды. Вместо того, чтобы вводить что-то вроде ‘`a \over b \over 2`’, надо указать, что над чем расположено:

`$${a\over b}\over 2$$`	$${a\over b}\over 2$$
`$$a\over{b\over 2}$$`	$$a\over{b\over 2}$$

К несчастью, обе эти альтернативы выглядят абсолютно ужасными. Математики, когда впервые печатают свою собственную работу с помощью такой системы типа TEX, имеют тенденцию “переиспользовать” `\over`. Хороший наборщик или редактор будет преобразовывать дроби в “строчную форму”, как только надстроенная конструкция окажется слишком маленькой или слишком тесной. Например, последние два случая могли быть выполнены следующим образом:

`$$a/b \over 2$$`	$$a/b\over 2$$

`$$a \over b/2$$` $\displaystyle\frac{a}{b/2}$

Преобразование в строчную форму требует некоторых математических знаний, поскольку иногда, чтобы сохранить значение формулы, требуется вставлять скобки. После подстановки `/` вместо `\over` обе части формулы должны быть заключены в скобки, если только они не простые символы. Например, $\frac{a}{b}$ превращается просто в a/b, но $\frac{a+1}{b}$ превращается в $(a+1)/b$, а $\frac{a+1}{b+1}$ — в $(a+1)/(b+1)$. Более того, обычно следует заключать в скобки целиком дробь, если она появляется рядом с чем-нибудь еще. Например, $\frac{a}{b}x$ превращается $(a/b)x$. Если у вас нет математического опыта, то в сомнительных случаях лучше попросить автора рукописи о помощи. Можно также тактично посоветовать ему в будущем избегать в рукописях дробей неприглядного вида.

▸ **Упражнение 17.1**
Как лучше всего получить формулу $x + y^{\frac{2}{k+1}}$?

▸ **Упражнение 17.2**
Преобразуйте в линейную форму $\frac{a+1}{b+1}x$.

▸ **Упражнение 17.3**
Какой сюрприз получил B. L. User, когда ввел `$$x = (y^2\over k+1)$$`?

▸ **Упражнение 17.4**
Как вы можете получить $7\frac{1}{2}¢$? (Предполагая, что команда `\cents` дает $¢$.)

Приведенные выше примеры показывают, что буквы и другие символы, когда появляются в дробях, иногда получаются меньшего размера, как они получаются меньшими, когда используются в качестве показателя степени. Настало время изучить, как TEX выбирает размеры символов. TEX имеет восемь стилей, в которых может обрабатывать формулы, а именно:

выделенный стиль	(для формул на отдельной строке)
текстовый стиль	(для формул в тексте)
индексный стиль	(в верхних или нижних индексах)
стиль повторных индексов	(в повторных индексах)

и четыре других “сжатых” стиля, которые почти такие же, за исключением того, что показатель степени не так поднят. Для краткости мы будем обозначать эти восемь стилей так:

$$D, D', T, T', S, S', SS, SS',$$

где D — выделенный стиль, D' — сжатый выделенный стиль, T — текстовый стиль и т.д. TEX также использует для математики три различных размера. Они называются: текстовый размер, размер индексов и размер повторных индексов.

Обычный способ ввести формулу с помощью TEX'а — заключить ее в знаки доллара `$` ... ,`$`. Это дает формулу в текстовом стиле (стиль T).

Или можно заключить ее в двойные знаки доллара `$$...$$` — это выделяет формулу в выделенном стиле (стиль D). Подформулы в этой формуле могут, конечно, быть в других стилях. Поскольку вы знаете стили, то можете определить размеры набора, которые TEX будет использовать:

Если буква в стиле	тогда она будет в	
D, D', T, T'	текстовом размере	(такая)
S, S'	размере индекса	(такая)
SS, SS'	размере повторного индекса	(такая)

Размера "SSS" — "размера повторного повторного индекса" — нет: такие крошечные символы были бы еще менее читабельными, чем размеры повторного индекса. Поэтому TEX оставляет размер повторного индекса наименьшим.

Стиль формулы	стиль верхнего индекса	стиль нижнего индекса
D, T	S	S'
D', T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Например, если `x^{a_b}` введено в стиле D, то `a_b` будет установлено в стиле S, а `b` — в стиле SS'; в результате будет x^{a_b}.

До сих пор мы не видели различия между стилями D и T. На самом деле существует легкое различие в положении показателя степени, хотя в каждом случае используется размер индекса. Вы получаете x^2 в стиле D, x^2 в стиле T и x^2 в стиле D или T — заметно различие? Но когда стили D и T переходят к дробям, то между ними большая разница:

Стиль формулы α`\over`β	стиль числителя α	стиль знаменателя β
D	T	T'
D'	T'	T'
T	S	S'
T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Таким образом, если вы вводите `1\over2` (в тексте), то получаете $\frac{1}{2}$, а именно, стиль S над стилем S'; но если вы вводите `$$1\over2$$`, то получаете

$$\frac{1}{2}$$

(выделенная формула), в которой стиль T над стилем T'.

На этом этапе мы могли бы так завершить правила стилей: `\underline` не меняет стиль, математические акценты и операции `\sqrt` и `\overline` меняют несжатые стили на их сжатые двойники; например, D изменяется на D', но D' остается прежним.

▸ **Упражнение 17.5**
Определите стиль и размер каждой части формулы $\sqrt{p_2^{e'}}$, предположив, что сама формула в стиле D.

Предположим, вам не нравится стиль, который TEX по своим правилам выбирает автоматически. Тогда вы можете задать нужный стиль, введя `\displaystyle`, `\textstyle`, `\scriptstyle` или `\scriptscriptstyle`. Стиль, который вы выбрали, будет применяться до конца формулы или подформулы или до тех пор, пока вы не выберете другой стиль. Например, `$$n+\scriptstyle n+\scriptscriptstyle n.$$` дает формулу

$$n+{\scriptstyle n+{\scriptscriptstyle n.}}$$

Это довольно глупый пример, но он показывает, что с изменением стиля знак плюс тоже становится меньше. TEX в индексных стилях не помещает пробелов вокруг +.

Приведем более полезный пример изменения стилей. Иногда вам надо напечатать цепную дробь, состоящую из последовательности других дробей, каждая из которых предполагается в выделенном стиле:

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{1}{a_4}}}}$$

Чтобы получить такой эффект, хорошо ввести

```
$$a_0+{1\over\displaystyle a_1+
        {\strut 1\over\displaystyle a_2+
          {\strut 1\over\displaystyle a_3+
            {\strut 1\over a_4}}}}$$
```

(Команда `\strut` использована, чтобы сделать знаменатель выше; это усовершенствование, которое будет обсуждаться в главе 18. А сейчас нас интересуют команды стиля.) Без `\strut` и `\displaystyle` в этой формуле результат был бы совершенно иной:

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{1}{a_4}}}}$$

Эти примеры показывают, что числитель и знаменатель дроби вообще говоря центрируются один относительно другого. Если вы предпочитаете, чтобы числитель или знаменатель были прижаты влево, поставьте после

них `\hfill`; а если вы предпочитаете, чтобы они были справа, поставьте `\hfill` слева. Например, если первые три `1\over` в предыдущем примере заменить на `1\hfill\over`, то получится

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4}}}}$$

(та форма цепной дроби, которую предпочитают многие авторы). Это работает потому, что `\hfill` растягивается в более сильной степени, чем клей, который используется TEX'ом, чтобы центрировать числитель и знаменатель.

У TEX'а есть операция `\atop`, которая аналогична `\over` за тем исключением, что опускает дробную черту:

```
$$x\atop y+2$$
```

$$x \atop y+2$$

В формате начального TEX'а в приложении B также определена команда `\choose`, которая аналогична `\atop`, только заключает результат в круглые скобки:

```
$$n\choose k$$
```

$$n \choose k$$

Она называется `\choose` (выбрать), потому что это — общая запись для так называемого биноминального коэффициента, который указывает, как много способов существует для выбора k объектов из n объектов.

Нельзя употреблять вперемежку `\over`, `\atop` и `\choose`. Например, `$$n \choose k \over 2$$` незаконно. Надо использовать группирование и вводить `$${n\choose k}\over2$$` либо `$$n\choose{k\over2}$$`, т.е.,

$$\frac{\binom{n}{k}}{2} \qquad \text{или} \qquad \binom{n}{\frac{k}{2}}.$$

Последняя формула, между прочим, выглядела бы лучше, если ее ввести как `$$n\choose k/2$$` или `$$n\choose{1\over2}k$$`, получая

$$\binom{n}{k/2} \qquad \text{или} \qquad \binom{n}{\frac{1}{2}k}.$$

▸ **Упражнение 17.6**

В качестве альтернативы $\frac{\binom{n}{k}}{2}$ обсудите, как можно получить две формулы:

$$\frac{1}{2}\binom{n}{k} \qquad \text{и} \qquad \frac{\binom{n}{k}}{2}.$$

▸ **Упражнение 17.7**

Объясните, как задать формулу:

$$\binom{p}{2}x^2y^{p-2} - \frac{1}{1-x}\frac{1}{1-x^2}.$$

TEX имеет обобщенную версию `\over` и `\atop`, в которой указывается точная толщина черты с помощью `\above`⟨размер⟩. Например,

```
$$\displaystyle{a\over b}\above1pt\displaystyle{c\over d}$$
```

напечатает составную дробь с усиленной (толщиной 1 pt) линейкой в качестве основной черты:

$$\frac{\frac{a}{b}}{\frac{c}{d}}.$$

Такие вещи встречаются в основном в учебниках по элементарной математике.

Математики часто используют для обозначения "суммы" знак $\sum$, а для обозначения "интеграла" — знак $\int$. Если вы наборщик, но не математик, вам надо запомнить, что `\sum` обозначает $\sum$, а `\int` — $\int$. Если вы забыли это, эти сокращения приводятся в приложении F вместе с другими символами. Символы типа $\sum$ и $\int$ (и несколько других символов типа $\bigcup$, $\prod$, $\oint$ и $\bigotimes$, перечисленных в приложении F) называются *большими операторами* и вводятся почти так же, как обычные символы или буквы. Отличие в том, что TEX в выделенном стиле выберет *больший* большой оператор, чем в текстовом стиле. Например,

`$\sum x_n$`	дает	$\sum x_n$	(T стиль)
`$$\sum x_n$$`	дает	$$\sum x_n$$	(D стиль).

Сумма в выделенном стиле обычно встречается с "пределами", т.е. с подформулами, которые печатаются над и под ней. Пределы вводятся так же, как если бы это были верхние и нижние индексы. Например, если вы хотите получить формулу

$$\sum_{n=1}^{m}$$

то введите либо `$$\sum_{n=1}^m$$`, либо `$$\sum^m_{n=1}$$`. В соответствии с обычными соглашениями о наборе математики, TEX заменит эту формулу на $\sum_{n=1}^m$ (т.е., без пределов), если предпочтительнее, чтобы она появилась в текстовом, а не в выделенном стиле.

Интегралы слегка отличаются от суммы тем, что в них даже в выделенном стиле верхние и нижние индексы не устанавливаются как пределы:

`$\int_{-\infty}^{+\infty}$`	дает	$\int_{-\infty}^{+\infty}$	(T стиль)

`$$\int_{-\infty}^{+\infty}$$` дает $\int_{-\infty}^{+\infty}$ (D стиль).

Некоторым наборщикам нравится ставить пределы над и под знаками $\int$. Это занимает больше места, но улучшает внешний вид, если подформула сложная, потому что отделяет пределы от остальной формулы. Аналогично, пределы изредка желательны в текстовом или индексном стиле, а некоторые наборщики предпочитают не ставить пределы и на выделенных знаках $\sum$. Можно изменить соглашения TEX'а, просто поставив `\limits` или `\nolimits` непосредственно после большого оператора. Например,

`$$\int\limits_0^{\pi\over2}$$` дает $\int\limits_0^{\frac{\pi}{2}}$

`$$\sum\nolimits_{n=1}^m$$` дает $\sum\nolimits_{n=1}^{m}$

Если вы говорите `\nolimits\limits` (например, потому что некоторые макрокоманды, также как `\int`, задают `\nolimits`, а вам нужны пределы), преимущество за последним словом. Имеется также команда `\displaylimits`, которую можно использовать, чтобы восстановить нормальные соглашения TEX'а, т.е., что пределы ставятся только в стилях D и D'.

Иногда под большим оператором нужно поместить два или более ряда пределов. Можно сделать это при помощи `\atop`. Например, если вы хотите получить выделенную формулу

$$\sum_{\substack{0\le i\le m\\ 0<j<n}} P(i,j)$$

то правильно это сделать так:

```
$$\sum_{\scriptstyle0\le i\le m\atop\scriptstyle0<j<n}P(i,j)$$
```

(возможно, еще с несколькими пробелами, чтобы в рукописном файле она выглядела более симпатично). Здесь инструкция `\scriptstyle` необходима дважды — иначе строки "$0 \le i \le m$" и "$0 < j < n$" будут в размере повторного индекса, который слишком мал. Это еще один пример редкого случая, когда автоматические правила TEX'а должны быть отвергнуты.

▸ **Упражнение 17.8**

Как бы вы ввели выделенную формулу $\sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r} a_{ij}b_{jk}c_{ki}$?

▸ **Упражнение 17.9**

А как бы вы обработали $\sum_{\substack{1\le i\le p\\ 1\le j\le q\\ 1\le k\le r}} a_{ij}b_{jk}c_{ki}$?

Поскольку математические формулы могут получаться ужасающе большими, TEX должен иметь какой-нибудь способ создавать все более увеличивающиеся символы. Например, если вы вводите

```
$$\sqrt{1+\sqrt{1+\sqrt{1+
          \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$
```

то результат показывает ряд имеющихся в наличии знаков квадратного корня:

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}$$

Три наибольших знака здесь, по существу, одинаковы, за исключением того, что вертикальный сегмент “ | ” повторяется столько, сколько необходимо, чтобы получить желаемый размер, но более маленькие знаки — это отдельные символы в математических шрифтах TEX’а.

Аналогичная вещь происходит со скобками и другими так называемыми “ограничивающими” символами. Например, приведем несколько различных размеров круглых и фигурных скобок, которые начальный TEX может использовать в формулах:

$$\left(\left(\left(\left(\left(\left(\left(\left(\,\right)\right)\right)\right)\right)\right)\right)\right)\left\{\left\{\left\{\left\{\left\{\left\{\left\{\left\{\,\right\}\right\}\right\}\right\}\right\}\right\}\right\}\right\}$$

Три наибольшие пары в каждом случае сделаны с повторяемым расширением, так что они могут стать сколь угодно большими.

Ограничители важны в математике, поскольку хорошо визуально подчеркивают структуры в сложных выражениях — они ограничивают отдельные подформулы. Приведем список 22 основных ограничителей, предусмотренных в начальном TEX’е.

Вход	*Ограничитель*
`(`	левая круглая скобка: (
`)`	правая круглая скобка:)
`[` или `\lbrack`	левая квадратная скобка: [
`]` или `\rbrack`	правая квадратная скобка:]
`\{` или `\lbrace`	левая фигурная скобка: {
`\}` или `\rbrace`	правая фигурная скобка: }
`\lfloor`	левая нижняя скобка: ⌊
`\rfloor`	правая нижняя скобка: ⌋
`\lceil`	левая верхняя скобка: ⌈
`\rceil`	правая верхняя скобка: ⌉
`\langle`	левая угловая скобка: ⟨

`\rangle`	правая угловая скобка: $\rangle$
`/`	косая черта: $/$
`\backslash`	обратная косая черта: $\backslash$
`|` или `\vert`	вертикальная черта: $\vert$
`\\|` или `\Vert`	двойная вертикальная черта: $\Vert$
`\uparrow`	верхняя стрелка: $\uparrow$
`\Uparrow`	двойная верхняя стрелка: $\Uparrow$
`\downarrow`	нижняя стрелка: $\downarrow$
`\Downarrow`	двойная нижняя стрелка: $\Downarrow$
`\updownarrow`	двусторонняя стрелка: $\updownarrow$
`\Updownarrow`	двойная двусторонняя стрелка: $\Updownarrow$

Иногда есть два способа получить один и тот же ограничитель. Например, можно задать левую квадратную скобку, введя либо `[`, либо `\lbrack`. Последнее предпочтительнее, поскольку `[` имеется не на всех клавиатурах. Помните, однако, что вы никогда не должны пытаться задавать левую и правую фигурные скобки просто как `{` или `}`. Символы `{` и `}` зарезервированы для группирования. Правильным будет ввести `\{`, `\}`, `\lbrace`, `\rbrace`.

Для того, чтобы получить несколько увеличенную версию любого из этих символов, просто поставьте перед ним `\bigl` (для открывающего ограничителя) или `\bigr` (для закрывающего ограничителя). Это облегчит чтение формул, которые содержат ограничители внутри ограничителей:

Вход	Выход
`$\bigl(x-s(x)\bigr)\bigl(y-s(y)\bigr)$`	$\bigl(x-s(x)\bigr)\bigl(y-s(y)\bigr)$
`$\bigl[x-s[x]\bigr]\bigl[y-s[y]\bigr]$`	$\bigl[x-s[x]\bigr]\bigl[y-s[y]\bigr]$
`$\bigl\| \|x\|+\|y\| \bigr\|$`	$\bigl\lvert\,\lvert x\rvert+\lvert y\rvert\,\bigr\rvert$
`$\bigl\lfloor\sqrt A\bigr\rfloor$`	$\bigl\lfloor\sqrt{A}\bigr\rfloor$

Ограничители `\big` настолько больше обычных, чтобы почувствовать различие, но еще достаточно малы, так что их можно использовать в тексте абзаца. Приведем все 22 ограничителя в обычном размере и в размере `\big`:

$$() [] \{\} \lfloor\rfloor \lceil\rceil \langle\rangle / \backslash \vert \Vert \uparrow\Uparrow\downarrow\Downarrow\updownarrow\Updownarrow$$
$$\big(\big) \big[\big] \big\{ \big\} \big\lfloor \big\rfloor \big\lceil \big\rceil \big\langle \big\rangle \big/ \big\backslash \big\vert \big\Vert \big\uparrow \big\Uparrow \big\downarrow \big\Downarrow \big\updownarrow \big\Updownarrow$$

Можно также использовать `\Bigl` и `\Bigr` для символов в выделенных формулах:

$$\Big(\Big) \Big[\Big] \Big\{ \Big\} \Big\lfloor \Big\rfloor \Big\lceil \Big\rceil \Big\langle \Big\rangle \Big/ \Big\backslash \Big\vert \Big\Vert \Big\uparrow \Big\Uparrow \Big\downarrow \Big\Downarrow \Big\updownarrow \Big\Updownarrow$$

Они на 50% выше их `\big`-двойников. Выделенные формулы чаще используют ограничители, которые еще выше (в два раза выше `\big`). Такие ограничители создаются при помощи `\biggl` и `\biggr` и выглядят так:

$$\bigg(\bigg) \bigg[\bigg] \bigg\{ \bigg\} \bigg\lfloor \bigg\rfloor \bigg\lceil \bigg\rceil \bigg\langle \bigg\rangle \bigg/ \bigg\backslash \bigg\vert \bigg\Vert \bigg\uparrow \bigg\Uparrow \bigg\downarrow \bigg\Downarrow \bigg\updownarrow \bigg\Updownarrow$$

Наконец, есть версия `\Biggl` и `\Biggr`, которая в 2.5 раза выше ограничителей `\bigl` и `\bigr`:

▸ **Упражнение 17.10**

Угадайте, как ввести формулу в $\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\left|\varphi(x+iy)\right|^2=0$ в выделенном стиле, используя для больших круглых скобок ограничитель `\bigg`. (Символы ∂ и φ, которые здесь присутствуют, вводятся с помощью `\partial` и `\varphi`.)

▸ **Упражнение 17.11**

На практике ограничители `\big` и `\bigg` используются намного чаще, чем `\Big` и `\Bigg`. Почему, как вы думаете, это правильно?

Ограничители `\bigl`, `\Bigl`, `\biggl` и `\Biggl` являются открывающими, как левая круглая скобка, а ограничители `\bigr`, `\Bigr`, `\biggr`, `\Biggr` — закрывающими, как правая круглая скобка. Начальный TEX также предусматривает ограничители `\bigm`, `\Bigm`, `\biggm` и `\Biggm` для использования в середине формул. Такой ограничитель играет роль отношения, типа знака равенства, поэтому TEX помещает с обеих сторон от него маленькие пробелы.

```
$\bigl(x\in A(n)\bigm| x\in B(n)\bigr)$
```
$\bigl(x\in A(n)\bigm| x\in B(n)\bigr)$

```
$\bigcup_n X_n\bigm\|\bigcap_n Y_n$
```
$\bigcup_n X_n\bigm\|\bigcap_n Y_n$

Можно также сказать просто `\big`, `\Big`, `\bigg`, `\Bigg`, что дает ограничитель, который действует как обычная переменная. Это используется преимущественно при наклонных чертах и обратных наклонных чертах, как показано в следующем примере:

```
$${a+1\over b}\bigg/{c+1\over d}$$
```
$${a+1\over b}\bigg/{c+1\over d}$$

▸ **Упражнение 17.12**

Как профессионально ввести $\bigl(x+f(x)\bigr)\big/\bigl(x-f(x)\bigr)$? (Будьте внимательны.)

TEX имеет встроенный механизм, который вычисляет, насколько высокой должна быть пара ограничителей для того, чтобы охватить данную подформулу, поэтому можно использовать этот метод вместо того, чтобы решать, должен ли быть ограничитель `\big`, `\bigg` или какой-нибудь еще. Единственное, что надо сделать — это сказать

`\left`⟨ограничитель$_1$⟩⟨подформула⟩`\right`⟨ограничитель$_2$⟩

и TEX напечатает подформулу, вставляя слева и справа заданные ограничители. Размер ограничителей будет как раз такой величины, чтобы охватить

подформулу. Например, в выделенной подформуле

```
$$1+\left(1\over1-x^2\right)^3$$
```

$$1+\left(\frac{1}{1-x^2}\right)^3$$

TEX выбрал `\biggl` (и `\biggr`), поскольку меньшие ограничители для этой дроби слишком малы. Простая формула типа `$\left(x\right)$` дает $\left(x\right)$, таким образом, `\left` и `\right` иногда выбирают ограничители, которые меньше, чем `\bigl` и `\bigr`.

Всякий раз, когда вы используете `\left` и `\right`, они должны быть в паре друг с другом, как и фигурные скобки в группах. Не может быть `\left` в одной формуле, а `\right` в другой, а также не позволяются конструкции типа

```
\left(...{...\right)...}
```

или

```
\left(...\begingroup...\right)...\endgroup.
```

Эти ограничения понятны, поскольку TEX'у надо набрать подформулу между `\left` и `\right` прежде, чем он может решить, насколько большими делать ограничители. Но это стоит ясно упомянуть, потому что вы *не обязаны* употреблять парами круглые, квадратные и тому подобные скобки, если не используете `\left` и `\right`: TEX не будет возражать против формулы `$[0,1)$`, `$)($` или даже `$)$`. (И хорошо, что TEX этого не делает для несбалансированных формул, которые удивительно часто встречаются в математических текстах.) Даже когда используется `\left` и `\right`, TEX не рассматривает подробно, какие конкретные ограничители выбраны. Так, можно напечатать такую странную штуку как "`\left)`" и/или "`\right(`", если вы понимаете, что делаете (и даже если не понимаете). Операция `\over` в примере выше, не включает в себя "`1+`" в начале формулы. Это получилось потому, что `\left` и `\right` выполняют функцию группирования в дополнение к функции создания ограничителя. Любые определения, которые оказываются между `\left` и `\right`, будут локальными, как если бы заключенная в них подформула была заключена в фигурные скобки.

▸ **Упражнение 17.13**
Используйте `\left` и `\right`, чтобы напечатать следующую выделенную формулу (с `\phi` для ϕ):

$$\pi(n) = \sum_{k=2}^{n} \left\lfloor \frac{\phi(k)}{k-1} \right\rfloor .$$

Сейчас вы, вероятно, удивляетесь, почему приходится тратить силы, изучая `\bigl`, `\bigr` и им подобные, когда `\left` и `\right` должны автоматически вычислять размеры. Да, это правда, `\left` и `\right` достаточно удобны, но есть как минимум три ситуации, когда для выбора размеров ограничителей вам придется воспользоваться собственной мудростью.

(1) Иногда `\left` и `\right` выбирают размеры меньше, чем вы хотите. Например, в одном из предыдущих примеров мы использовали `\bigl` и `\bigr`, чтобы получить $\big||x|+|y|\big|$; `\left` и `\right` не делают ничего сверх необходимого, так что `$\left|\left|x\right|+\left|y\right|\right|$` дает всего лишь $||x|+|y||$' (2) Иногда `\left` и `\right` выбирают размеры больше, чем вы хотите. Это наиболее часто случается, когда они ограничивают большой оператор в выделенной формуле. Например, сравните две следующие формулы:

```
$$\left( \sum_{k=1}^n A_k \right)$$
```
$$\left(\sum_{k=1}^n A_k \right)$$

```
$$\biggl( \sum_{k=1}^n A_k \biggr)$$
```
$$\biggl(\sum_{k=1}^n A_k \biggr)$$

Правила для `\left` и `\right` говорят охватить `\sum` вместе с ее пределом,но в некоторых специальных случаях лучше позволить пределам немного высовываться. Здесь лучше применить ограничитель `\bigg`. (3) Иногда надо разбить огромную выделенную формулу на две или более отдельные строки, и вы хотите быть уверенными, что открывающиеся и закрывающиеся ограничители имеют одинаковую величину. Но вы не можете использовать на первой строке `\left`, а на последней — `\right`, поскольку `\left` и `\right` должны встречаться парами. Решением будет использовать `\Biggl` (скажем) на первой строке и `\Biggr` — на последней.

Конечно, одним из преимуществ `\left` и `\right` является то, что они могут сделать произвольно большие ограничители — намного больше, чем `\biggggg`! Однако косые черты и угловые скобки имеют максимальный размер. Если вы потребуете более крупные варианты этих символов, то получите наибольший из возможных.

▸ **Упражнение 17.14**

Докажите, что вы овладели ограничителями: заставьте TEX выдать формулу

$$\pi(n) = \sum_{m=2}^{n} \left\lfloor \left(\sum_{k=1}^{m-1} \lfloor (m/k)/\lceil m/k \rceil \rfloor \right)^{-1} \right\rfloor .$$

Если ввести '`.`' после `\left` или `\right` вместо одного из основных ограничителей, то получится так называемый нулевой ограничитель (который равен пробелу). Кому это может понадобиться? Ну, иногда нужны формулы, которые содержат только один большой ограничитель. Например, выделенная формула

$$|x| = \begin{cases} x, & \text{если } x \ge 0 \\ -x, & \text{если } x < 0 \end{cases}$$

имеет '{', но не имеет '}'. Она может быть получена конструкцией вида

```
$$|x|=\left\{ ... \right.$$
```

Глава 18 объясняет, как заполнить `...`, чтобы завершить эту конструкцию; сейчас только отметим, что `\right.` даст возможность получить невидимый правый ограничитель, соответствующий левой фигурной скобке.

Нулевой ограничитель не абсолютно пуст — это пустой бокс, ширина которого равна параметру TEX'а, называемому `\nulldelimiterspace`. Мы позже увидим, что нулевые ограничители вставляются вслед за дробями. Начальный TEX устанавливает `\nulldelimiterspace=1.2pt`.

Можно вводить `<` или `>` как условное сокращение для `\langle` и `\rangle`, когда TEX ищет ограничитель. Например, `\bigl<` эквивалентно `\bigl\langle`, а `\right>` эквивалентно `\right\rangle`. Конечно, '`<`' и '`>`' обычно производят отношения "меньше чем" и "больше чем" $<>$, которые заметно отличаются от угловых скобок $\langle\rangle$.

Начальный TEX имеет и дополнительные ограничители, которые не были перечислены в основном наборе 22 ограничителей, потому что они особого сорта. Команды `\arrowvert`, `\Arrowvert` и `\bracevert` производят ограничители, сделанные из повторяющихся частей вертикальных стрелок, двойных вертикальных стрелок и больших фигурных скобок, соответственно, без верхушки стрелок и без закрученной части фигурных скобок. Результат аналогичен `\vert` или `\Vert`, но у них большие пробелы и другая ширина. Также можно использовать `\lgroup` и `\rgroup`, которые сконструированы из фигурных скобок без их средней части, и `\lmoustache` и `\rmoustache`, которые дают верхние и нижние половины фигурных скобок. Например, приведем `\Big` и `\bigg` версии от `\vert`, `\Vert` и этих семи специальных ограничителей:

$$\ldots\Big|\ldots\Big\|\ldots\Big|\ldots\Big\|\ldots\Big\lgroup\ldots\Big\rgroup\ldots\Big\lmoustache\ldots\Big\rmoustache\ldots;$$
$$\ldots\bigg|\ldots\bigg\|\ldots\bigg|\ldots\bigg\|\ldots\bigg\lgroup\ldots\bigg\rgroup\ldots\bigg\lmoustache\ldots\bigg\rmoustache\ldots.$$

Заметим, что `\lgroup` и `\rgroup` довольно похожи на жирные круглые скобки с острыми изгибами по углам. Это делает их заманчивыми для некоторых больших выделенных формул. Но их нельзя использовать точно так же, как круглые скобки, потому что они доступны только в больших размерах (`\Big` и больше).

Вопрос: что случиться, если за большим ограничителем следует верхний или нижний индекс? Ответ: это хороший вопрос. После ограничителя `\left` — это первый верхний или нижний индекс ограниченной подформулы, ибо в действительности ему предшествует `{}`. После ограничителя `\right` — это верхний или нижний индекс целой `\left...\right` подформулы. А после ограничителей `\bigl`, `\bigr`, `\bigm` или `\big` он применяется только к этому отдельному ограничителю. Таким образом, '`\bigl(_2`' работает совсем не так, как '`\left(_2`'.

Если вы внимательно посмотрите на примеры математического набора, приведенные в этой главе, вы заметите, что большие круглые и квадратные скобки симметричны относительно невидимой горизонтальной линии, которая проходит немного выше базовой линии. Когда ограничитель увеличивается, его высота и глубина увеличиваются на одну и ту же величину. Эта горизонтальная линия называется *осью* формулы; например, формула в тексте этого абзаца имела

бы ось на таком уровне: ——. Дробная черта каждой дроби центрирована на оси безотносительно к величине числителя и знаменателя.

Иногда необходимо создать специальный бокс, который должен быть центрирован вертикально относительно оси. (Например, пример '$|x| = \{\ldots$', приведенный выше, был сделан с применением такого бокса.) TEX для этого имеет простой способ: вы просто говорите

`\vcenter{`⟨вертикальный материал⟩`}`

и вертикальный материал будет упакован в бокс так же, как если бы вместо `\vcenter` был `\vbox`. Затем бокс будет подниматься или опускаться до тех пор, пока его верхняя граница не будет на столько же выше оси, на сколько нижняя граница ниже ее.

Понятие "ось" значимо для TEX'а только в математических формулах, но не в обычном тексте, поэтому TEX позволяет использовать `\vcenter` только в математической моде. Если вам все же надо центрировать что-нибудь вертикально в горизонтальной моде, правильно будет сказать `$\vcenter{...}$`. (Между прочим, конструкции `\vcenter to`⟨размер⟩ и `\vcenter spread`⟨размер⟩ в математической моде тоже законны; вертикальный клей всегда устанавливается по правилам для `\vbox`, приведенным в главе 12. Но обычно достаточно одного `\vcenter`.)

В формулу можно вставить любой бокс, просто обычным образом сказав `\hbox`, `\vbox`, `\vtop`, `\box` или `\copy`, даже когда вы находитесь в математической моде. Более того, можно использовать `\raise` или `\lower`, как в горизонтальной моде, а также вертикальные линейки при помощи `\vrule`. Конструкции типа `\vcenter` производят боксы, которые в математических формулах могут быть использованы как обычные символы.

Иногда, встречаясь с чем-либо необычным, чего нет в шрифтах, вы захотите создать свои собственные символы. Если новый символ встречается только в одном месте, можно использовать `\hbox`, `\vcenter` или еще что-нибудь, чтобы вставить в точности то, что вам надо. Но если вы определяете макрокоманду для общего использования, можно использовать различные конструкции в различных стилях. TEX имеет специальную возможность, называемую `\mathchoice`, которая в этом приходит вам на помощь. Вы пишите

`\mathchoice{`⟨мат⟩`}{`⟨мат⟩`}{`⟨мат⟩`}{`⟨мат⟩`}`

где каждая ⟨мат⟩ указывает подформулу. TEX будет выбирать первую подформулу в стиле D или D', вторую в стиле T или T', третью — в стиле S или S', а четвертую — в стиле SS или SS'. (TEX создает макеты всех четырех подформул, прежде чем выберет окончательный вариант, потому что, когда встречается `\mathchoice`, действительный стиль не всегда известен. Например, когда вы печатаете `\over`, вы часто меняете стиль всего, что встретилось в формуле до этого. Поэтому `\mathchoice` довольно дорог в смысле времени и места, и использовать его можно только тогда, когда вы согласны за это платить.)

Упражнение 17.15
Догадайтесь, что будет получено следующими командами:

```
\def\puzzle{{\mathchoice{D}{T}{S}{SS}}}
$$\puzzle{\puzzle\over\puzzle^{\puzzle^\puzzle}}$$
```

Упражнение 17.16
Придумайте макрокоманду `\square`, которая делает □ для использования в математических формулах. Бокс должен быть симметричным относительно оси, а его внутренние размеры должны быть 3 pt в выделенном и текстовом стилях, 2.1 pt в индексовом стилях и 1.5 pt в стиле повторных индексов. Линейки должны быть толщиной 0.4 pt в выделенном и текстовом стилях и 0.3 pt в других стилях.

Начальный TEX имеет макрокоманду `\mathpalette`, которая полезна для конструкций `\mathchoice`; `\mathpalette\a{xyz}` раскрывается в ряд из четырех вариантов:

```
\mathchoice{\a\displaystyle{xyz}}...{\a\scriptscriptstyle{xyz}}.
```

Таким образом, первый аргумент — `\mathpalette` — это команда, первый аргумент которой — выбор стиля. Приложение B содержит несколько примеров, которые показывают, как применяется `\mathpalette`. (В частности, определения `\phantom`, `\root` и `\smash`; знак конгруенции `\cong` ($\cong$) также создан из $=$ и $\sim$, используя `\mathpalette`.)

В начале этой главы мы обсуждали команды `\over`, `\atop`, `\choose` и `\above`. Они являются специальным случаем понятия “обобщенной дроби” TEX'а, которая включает также три примитива.

```
\overwithdelims⟨ограничитель1⟩⟨ограничитель2⟩
\atopwithdelims⟨ограничитель1⟩⟨ограничитель2⟩
\abovewithdelims⟨ограничитель1⟩⟨ограничитель2⟩⟨размер⟩
```

Третий случай наиболее общий, т.к. заключает в себе все другие обобщенные дроби: `\overwithdelims` использует дробную черту, толщина которой принята по умолчанию для текущего размера,`\atopwithdelims` использует невидимую дробную черту, толщина которой равна нулю, в то время как `\abovewithdelims` использует дробную черту, толщина которой указана явно. TEX помещает подформулу ⟨числитель⟩ над подформулой ⟨знаменатель⟩, отделенной дробной чертой заданной толщины; затем он ставит ⟨ограничитель$_1$⟩ слева и ⟨ограничитель$_2$⟩ справа. Например, `\choose` эквивалентно `\atopwithdelims()`. Если вы определяете `\legendre` как `\overwithdelims()`, то можете печатать символ Лежандра $\left(\frac{a}{b}\right)$, сказав `{a\legendre b}`. Размер окружающих ограничителей зависит только от стиля, а не от размера дробей. Большие ограничители используются в стилях D и D' (см. приложение G). Простые команды `\over`, `\atop` и `\above` эквивалентны соответствующим ‘`withdelims`’-командам, ограничители в которых нулевые. Например, `\over`— это сокращение для `\overwithdelims...`

Упражнение 17.17
Определите команду `\euler` так, чтобы в формуле `$n\euler k$` получилось число Эйлера $\left\langle {n \atop k} \right\rangle$.

Приложение G объясняет точно, как TEX вычисляет требуемый размер ограничителей для `\left` и `\right`. Общая идея состоит в том, что ограничители вертикально центрированы относительно оси. Следовательно, если вы хотите охватить подформулу между `\left` и `\right`, в которой над осью располагаются y_1 единиц, а под осью — y_2 единиц, вам надо сделать ограничитель, высота плюс глубина которого равны как минимум y , где $y = 2\max(y_1, y_2)$. Обычно лучше не полностью охватить формулу, а подойти к этому близко, поэтому TEX позволяет задать два параметра, `\delimiterfactor` f (целое) и `\delimitershortfall` δ (размер). Минимальный размер ограничителя выбирается так, чтобы быть не меньше $y \cdot f/1000$ и не меньше $y - \delta$. Приложение B устанавливает $f = 901$ и $\delta = 5\,\mathrm{pt}$. Таким образом, если $y = 30\,\mathrm{pt}$, формат начального TEX'а указывает, что высота ограничителя должна быть больше 27 pt, а если $y = 100\,\mathrm{pt}$, соответствующий ограничитель будет иметь размер как минимум 95 pt.

До сих пор мы обсуждали правила для ввода математических формул, но ничего не сказали о том, как TEX занимается преобразованием своих входных данных в списки из боксов и клея. Почти все команды, которые были упомянуты в главах 16 и 17 — это элементы "высокого уровня" формата начального TEX'а. Они не встроены в сам TEX. Приложение B определяет такие команды в терминах более примитивных команд, с которыми TEX реально имеет дело. Например, `\choose` — это сокращение для `\atopwithdelims()`. Приложение B не только вводит `\choose`, но также сообщает TEX'у, где найти ограничители '`(`' и '`)`' в различных размерах. Формат начального TEX'а определяет все специальные символы типа `\alpha` и `\mapsto`, все специальные акценты типа `\tilde` и `\widehat`, все большие операторы типа `\sum` и `\int`, и все ограничители типа `\lfloor` и `\vert`. Любой из них можно переопределить, чтобы адаптировать TEX к другим математическим стилям и/или к другим шрифтам.

Оставшаяся часть этой главы обсуждает команды низшего уровня, которые руководят TEX'ом из-за кулис. Каждый абзац следующих нескольких страниц отмечен двойным знаком опасного поворота, поэтому, если вы не жаждете TEXнических подробностей, то можете перескочить на главу 18.

Все символы, которые печатаются в математической моде, принадлежат к одному из шестнадцати *семейств шрифтов*, пронумерованных от 0 до 15. Каждое из этих семейств состоит из трех шрифтов: один для текстового размера, один для индексного размера и один для размера повторного индекса. Для того, чтобы определить члены каждого семейства, используются команды `\textfont`, `\scriptfont` и `\scriptscriptfont`. Например, в формате начального TEX'а семейство 0 используется для букв прямого шрифта, и для задания этого семейства приложение B содержит инструкции

```
\textfont0=\tenrm
\scriptfont0=\sevenrm
\scriptscriptfont0=\fiverm
```

Прямой шрифт в 10 пунктов (`\tenrm`) используется для обычных символов, прямой шрифт в 7 пунктов (`\sevenrm`) — для индексов, а прямой шрифт в 5 пунктов (`\fiverm`) — для повторных индексов. Поскольку в каждом шрифте до 256 символов, в семействе по 3 шрифта, а всего 16 семейств, TEX'у в любой формуле доступны до 12288 символов (по 4096 в каждом из трех размеров)

Определения типа `\textfont`⟨номер семейства⟩`=`⟨идентификатор шрифта⟩ являются локальными в группе, которая их содержит, так что можно легко изменить принадлежность семейству из одного набора соглашений на другое и вернуться обратно. Более того, можно поместить любой шрифт в любое семейство. Например, команда

```
\scriptscriptfont0=\scriptfont0
```

делает величину нижних повторных индексов в семействе 0 такой же, как текущая величина нижних индексов. TEX не задумывается над тем, разумно ли организованы семейства, он просто следует инструкциям. (Однако, как мы увидим позже, шрифты не могут быть использованы в семействах 2 и 3, если они не содержат некоторого числа специальных параметров.) Между прочим, TEX для каждого неопределенного члена семейства использует `\nullfont`, — шрифт, который не содержит символов.

В то время, когда читается математическая формула, TEX запоминает каждый символ, как "символ в такой-то такой-то позиции семейства номер такой-то — такой-то", но он не замечает, какие шрифты в действительности находятся в семействах до того, как достигает конца формулы. Таким образом, если вы загрузили шрифт, называемый `\Helvetica`, который содержит швейцарские цифры, и если вы скажете что-то типа

```
$\textfont0=\tenrm 9 \textfont0=\Helvetica 9$
```

то получите две девятки в шрифте `\Helvetica`, предполагая, что TEX'у задано брать девятки из семейства 0. Причина в том, что `\textfont0` равен `\Helvetica` в конце формулы, а так же когда она рассматривается. С другой стороны, если вы скажете

```
$\textfont0=\tenrm 9 \hbox{$9\textfont0=\Helvetica$}$
```

то первая 9 будет из `\tenrm`, а вторая из `\Helvetica`, потому что формула в h-боксе будет набрана прежде, чем включена в окружающую формулу.

▸ **Упражнение 17.18**
Если вы введете `${\textfont0=\Helvetica 9}$`, то какой шрифт будет использован для 9?

Каждому математическому символу задано идентифицирующее кодовое число от 0 до 4095, которое получается, если добавить к номеру позиции 256 , умноженное на номер семейства. Это легко выражается в шестнадцатиричной записи, используя одну шестнадцатиричную цифру для семейства и две — для символа. Например, `"24A` обозначает символ `"4A` семейства 2. Каждый символ отнесен к одному из следующих восьми классов, пронумерованных от 0 до 7.

Класс	*Значение*	*Пример*
0	Обычные	`/`
1	Большие операторы	`\sum`
2	Бинарные операторы	`+`
3	Отношения	`=`
4	Открывающие	`(`

5	Закрывающие	`)`
6	Пунктуация	`,`
7	Переменное семейство	`x`

Классы с 0 до 6 указывают на "части речи", которые принадлежат математическому языку, класс 7 — это специальный случай, который обсуждается ниже. Номер класса умножается на 4096 и прибавляется к номеру символа, а это то же самое, что сделать его первой цифрой четырехзначного шестнадцатиричного числа. Например, приложение B определяет, что `\sum` — это математический символ `"1350`, а это означает — большой оператор (класс 1), находящийся в позиции `"50` семейства 3.

Упражнение 17.19
Символы `\oplus` и `\bullet` ($\oplus$ и $\bullet$) — это бинарные операции, которые появляются в позициях 8 и 15 (десятичной) семейства 2, когда используются шрифты начального TEX'а. Угадайте, каков их математический код. (Это очень легко.)

Класс 7 — это специальный случай, который позволяет математическим символам изменять семейство. Он ведет себя точно так же, как класс 0, за исключением того, что указанное семейство заменяется на текущее значение целого параметра, называемого `\fam`, при условии, что `\fam` — это законный номер семейства (т.е., если он лежит между 0 и 15). TEX, когда входит в математическую моду, автоматически устанавливает `\fam=-1`, поэтому классы 7 и 0 эквивалентны, если только `\fam` не присвоено новое значение. Начальный TEX изменяет `\fam` на 0. когда пользователь вводит `\rm`. Это делает удобным получать в формулах буквы прямого шрифта, как мы это увидим в главе 18, поскольку буквы принадлежат классу 7. (Команда `\rm` — это сокращение для `\fam=0 \tenrm`; таким образом, `\rm` указывает `\fam` стать нулевым и делает `\tenrm` "текущим шрифтом". В горизонтальной моде значение `\fam` не действует и буквы печатаются под управлением текущего шрифта, а в математической моде к делу не относится текущий шрифт и буквы управляются значением `\fam`. Текущий шрифт действует на математическую моду только если использован `\␣` или если размеры выражены в единицах `ex` или `em`. Он также оказывает воздействие, если внутри формулы появляется `\hbox`, поскольку содержимое h-бокса печатается в горизонтальной моде.)

Интерпретация символов в математической моде определяется таблицей 128 значений "математических кодов". Элементы этой таблицы могут быть изменены командой `\mathcode` точно также, как номера категории изменяются командой `\catcode` (см. главу 7). Каждый математический код указывает класс, семейство и положение символа, как это было описано выше. Например, приложение B содержит команды

```
\mathcode`<="313C
\mathcode`*="2203
```

которые указывают TEX'у трактовать символ `<` в математической моде как отношение (класс 3), расположенное в позиции `"3C` семейства 1, а звездочку `*` — как бинарную операцию в позиции 3 семейства 2. Начальное значение `\mathcode`b` равно `"7162`. Таким образом, `b` — это символ `"62` в семействе 1 (курсив), а его семейство может быть изменено при помощи `\fam`. TEX смотрит на математический код только тогда, когда печатает символы с номером категории 11 (буква)

или 12 (другие), или когда ему встречается символ, заданный явным образом как `\char`⟨число⟩. (Если `\char` используется с символьным кодом между 128 и 255, то для `\mathcode` нет числового значения. В этом случае подразумевается семейство 0 и класс 0.

`\mathcode` может также иметь специальное значение `"8000`, которое заставляет символ вести себя так, как если бы у него был номер категории 13 (активный). Приложение B использует эту особенность, чтобы раскрыть `'` в `^{\prime}` довольно хитрым способом. Математический код `'` не может использоваться вместе с `'` в восьмеричных константах.

Таблица математических кодов позволяет косвенно ссылаться на любой символ любого семейства. Нажав одну клавишу, можно задать математический код символа непосредственно, напечатав `\mathchar`, который аналогичен `\char`. Например, команда `\mathchar"1ABC` задает символ класса 1 семейства 10 (`"A`) позиции `"BC`. Поэтому достаточно около ста определений типа

```
\def\sum{\mathchar"1350 }
```

чтобы определить специальные символы начального TEX'a. Но существует лучший способ: TEX имеет примитивную команду `\mathchardef`, которая относится к `\mathchar` так же, как `\chardef` относится к `\char`. Приложение B содержит около ста определений типа

```
\mathchardef\sum="1350
```

чтобы определить математические символы. `\mathchar` должен быть между 0 и 32767 (`"7FFF`).

Символ класса 1, т.е., большой оператор типа `\sum`, будет при печати вертикально центрироваться относительно оси. Таким образом, большие операторы могут быть использованы с различными размерами шрифтов. Для символов других классов вертикальное регулирование не делается.

TEX связывает классы с подформулами так же, как с отдельными символами. Так, например, вы можете трактовать составную конструкцию, как если бы она была бинарной операцией или отношением. Для этой цели используются команды `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose` и `\mathpunct`. За каждой из них следует либо простой символ либо подформула, заключенная в фигурные скобки. Например, `\mathopen\mathchar"1234` эквивалентно `\mathchar"4234`, потому что `\mathopen` вставляет класс 4 (открывание). В формуле `$G\mathbin:H$` двоеточие рассматривается как бинарная операция. А приложение B конструирует большие открывающие символы, определяя `\bigl#1` как сокращение для

```
\mathopen{\hbox{$\left#1 ...\right.$}}
```

Существует также восьмая классификация — `\mathinner`, которая обычно не используется для ординарных символов. Дроби и конструкции `\left...\right` рассматриваются как "внутренние" подформулы, а это означает, что в некоторых обстоятельствах они не окружаются дополнительными пробелами. Все другие подформулы обычно трактуются как обыкновенные символы, независимо от того,

сформированы они при помощи `\overline`, `\hbox`, `\vcenter` или просто заключены в фигурные скобки. Таким образом, `\mathord` в действительности не является необходимой частью языка TEX. Вместо того, чтобы писать `$1\mathord,234$`, можно получить тот же самый эффект от `$1{,}234$`.

▸ **Упражнение 17.20**
Команды типа `\mathchardef\alpha="10B` использованы в приложении B для определения строчных греческих букв. Предположим, вы хотите расширить начальный TEX, поместив жирные математические курсивные буквы в семейство 9 аналогично обычным математическим курсивным буквам в семействе 1. (Таких шрифтов нет в кратких версиях TEX'а, но давайте предположим, что они существуют.) Предположим, команда `\bmit` определена как сокращение для `\fam=9`, следовательно, `{\bmit b}` дает жирное математическое курсивное `b`. Какое изменение в определении `\alpha` надо сделать, чтобы `{\bmit\alpha}` производила жирную альфу?

Ограничители заданы похожим, но более сложным способом. Каждый символ имеет не только `\catcode` и `\mathcode`, но также `\delcode`, который либо отрицательный (для символов, которые не могут действовать как ограничители), либо меньше `"1000000`. Другими словами, неотрицательные ограничительные коды состоят из шести шестнадцатиричных цифр. Первые три цифры специфицируют "маленький" вариант ограничителя, последние три — "большой". Например, команда

```
\delcode`x="123456
```

означает, что если буква `x` используется как ограничитель, то ее маленький вариант расположен в позиции `"23` семейства 1, а большой вариант — в позиции `"56` семейства 4. Однако, если большой или маленький вариант задан как `000`, (позиция 0 семейства 0), то этот вариант игнорируется. TEX смотрит на ограничительный код, когда символ следует за `\left`, `\right` или за одной из команд `withdelims`; отрицательный ограничительный код ведет к сообщению об ошибке, иначе TEX ищет подходящий ограничитель, попробовав сначала маленький вариант, затем большой. (Приложение G более детально обсуждает этот процесс.) Например, приложение B содержит команды

```
\delcode`(="028300  \delcode`.=0
```

которые указывают, что маленький вариант левой круглой скобки расположен в позиции `"28` семейства 0, а большой — в позиции 0 семейства 3. Так как точка не имеет вариантов, `\left.` даст нулевой ограничитель. В действительности в семействе 3 существует несколько различных символов круглых скобок. Наименьшая находится в позиции 0, а другие связаны вместе информацией, которая поступает со шрифтом. Все ограничительные коды равны -1, пока они не изменены командой `\delcode`.

▸ **Упражнение 17.21**
Приложение B определяет `\delcode`<` так, что это — сокращенная запись для угловых скобок. Почему, как вы думаете, приложение B не идет дальше и не определяет `\delcode`{`?


Ограничители также могут быть заданы как `\delimiter`⟨число⟩. В этом случае число может быть так же велико, как `"7FFFFFF`, т.е., семь шестнадцатиричных цифр. Ведущая цифра указывает класс от 0 до 7, как в `\mathchar`. Например, приложение B содержит определение

```
\def\langle{\delimiter"426830A }
```

Это означает, что `\langle` — это открывающийся символ (класс 4), маленький вариант которого `"268`, а большой — `"30A`. Когда `\delimiter` появляется после `\left` или `\right`, цифра класса игнорируется , но когда `\delimiter` встречается в других контекстах, т.е. когда TEX не ищет ограничитель, три правые цифры опускаются, а оставшиеся четыре цифры действуют как `\mathchar`. Например, выражение `$\langle x$` трактуются, как если бы это было `$\mathchar"4268 x$`.


▸ **Упражнение 17.22**
Что получится неправильно, если вы введете такую команду:

```
bigl\delimiter"426830A thinspace?
```


Конечно, эти числовые соглашения для `\mathchar` и `\delimiter` не слишком красивы, но они позволяют надежно упаковывать множество информации в маленьком пространстве. Вот почему TEX использует их для определений низшего уровня внутри форматов. Также заслуживают упоминания два других примитива низшего уровня: `\radical` и `\mathaccent`. Начальный TEX делает доступными знак квадратного корня и математические акценты, задавая команды

```
\def\sqrt{\radical"270370 }
\def\widehat{\mathaccent"362 }
```

и еще несколько таких же. Идея этого заключается в том, что за `\radical` следует код ограничителя, а за `\mathaccent` — код математического символа, так что TEX знает семейство и символьные позиции для символов, используемых в конструкциях с радикалами и акцентами. Приложение G дает точную информацию о расположении этих символов. Изменяя определения, можно легко расширять TEX так, чтобы он печатал различные знаки радикала и различал знаки акцентов, если только такие символы имеются в шрифтах.


Начальный TEX использует семейство 1 для математических курсивных букв, семейство 2 — для обычных математических символов, а семейство 3 — для больших символов. TEX настаивает, чтобы шрифты в семействах 2 и 3 имели специальные параметры `\fontdimen`, которые управляли бы математическими пробелами в соответствии с правилами приложения G. Символьные шрифты `cmsy` и `cmex` имеют эти параметры, поэтому их принадлежность к семействам 2 и 3 почти обязательна. (Существует, однако, способ изменить параметры любого шрифта, используя команду `\fontdimen` .) `INITEX` инициализирует математические коды всех букв от `A` до `Z` и от `a` до `z` так, что все они являются символами класса 7 семейства 1; вот почему естественно использовать семейство 1 для математического курсива. Аналогично, цифры от `0` до `9` принадлежат классу 7 и семейству 0. Никакое из других семейств не трактуется TEX'ом специальным образом. Так, например, начальный TEX помещает текстовый курсив в семейство 4, наклонные буквы — в семейство 5, жирные прямые буквы — в семейство 6, шрифт

пишущей машинки — в семейство 7, но любое из этих чисел может быть переключено. Существует макрокоманда `\newfam`, аналогичная `\newbox`, присваивающая символические имена семействам, которые еще не использовались.

Когда TEX находится в горизонтальной моде, он создает горизонтальный список, в вертикальной моде он создает вертикальный список. Поэтому не будет большим сюрпризом, что в математической моде TEX создает математический список. Содержание горизонтального списка объяснялось в главе 14, содержание вертикального списка — в главе 15. Сейчас настало время описать, из чего состоят математические списки. Каждый элемент в математическом списке принадлежит к одному из следующих типов.

- атом (объясняется здесь же);
- горизонтальный материал (линейка, discretionary, штраф или "whatsit");
- вертикальный материал (из `\mark`, `\insert` или `\vadjust`);
- порция клея (из `\hskip`, `\mskip` или `\nonscript`);
- керн (из `\kern` или `\mkern`);
- изменение стиля (из `\displaystyle`, `\textstyle`, и т.д);
- обобщенная дробь (из `\above`, `\over`, и т.д);
- граница (из `\left` или `\right`);
- выбор из четырех путей (из `\mathchoice`).

Наиболее важные элементы называются *атомами*. Они имеют три части: *ядро*, *верхний индекс*, и *нижний индекс*. Например, если вы вводите

```
(x_i+y)^{\overline{n+1}}
```

в математической моде, то получаете математический список, состоящий из атомов: (, x_i, $+$, y, и $)^{\overline{n+1}}$. Ядрами этих атомов являются (, x, $+$, y, и); нижние индексы пустые, за исключением второго атома, который имеет нижний индекс i; верхние индексы пусты за исключением последнего атома, чей верхний индекс равен $\overline{n+1}$. Этот верхний индекс сам есть математический список, состоящий из одного атома, чье ядро — это $n+1$, и это ядро есть математический список, состоящий из трех атомов.

Существует тринадцать видов атомов, каждый из которых в формуле может действовать по–разному. Например, '('— это атом Open, поскольку он появляется из открывающего символа. Приведем полный список различных видов атомов:

Ord	обычный атом, типа x;
Op	атом большого оператора типа $\sum$;
Bin	атом бинарной операции типа $+$;
Rel	атом отношения, типа $=$;
Open	открывающий атом, типа $($;
Close	закрывающий атом, типа $)$;
Punct	атом пунктуации, типа $,$;
Inner	внутренний атом, типа $\frac{1}{2}$;
Over	подчеркнутый атом, типа $\overline{x}$;
Under	подчеркнутый атом, типа $\underline{x}$;
Acc	акцентированный атом, типа $\hat{x}$;

Rad атом радикала, типа $\sqrt{2}$;
Vcent центрированный вертикальный бокс, полученный при помощи `\vcenter`.

Ядра, верхние и нижние индексы атома называются его *полями*, и для каждого из этих полей есть четыре возможности: поле может быть

- пустым;
- математическим символом (заданным семейством и номером позиции);
- боксом;
- математическим списком.

Например, Close–атом '$)^{\overline{n+1}}$', рассмотренный выше, имеет пустое поле нижнего индекса, его ядро — символ ')' семейства 0 позиции `"28`, если действуют соглашения начального TEX'а; а его поле верхнего индекса является математическим списком $\overline{n+1}$. Последний математический список состоит из Over-атома, чье ядро — это математический список '$n+1$', а этот математический список, в свою очередь, состоит из трех атомов типа Ord, Bin, Ord.

Можно посмотреть, какой вид в TEX'е имеет математический список, введя в математической моде `\showlists`. Например, если вы введете `$(x_i+y)^{\overline{n+1}}\showlists`, ваш протокольный файл получит следующие любопытные данные:

```
\mathopen
.\fam0 (
\mathord
.\fam1 x
_\fam1 i
\mathbin
.\fam0 +
\mathord
.\fam1 y
\mathclose
.\fam0 )
^\overline
^.\mathord
^..\fam1 n
^.\mathbin
^..\fam0 +
^.\mathord
^..\fam0 1
```

В наших предыдущих опытах с `\showlists` мы наблюдали, что могут быть боксы внутри боксов, и что каждая строка в протокольном файле предваряется точками, чтобы указать ее положение в иерархии. Математические списки имеют намного более сложную структуру, поэтому точка используется для обозначения ядра атома, '`^`"""' — для поля верхнего индекса, а '`_`' — для поля нижнего индекса. Пустые поля не показаны. Так, например, Ord-атом 'x_i' представлен здесь тремя строчками '`\mathord`', '`.\fam1 x`' и '`_\fam1 i`'.

Некоторые виды атомов кроме полей ядер, верхних и нижних индексов несут дополнительную информацию: Op-атом будет помечен `\limits` или `\nolimits`, если подавлено обычное соглашение `\displaylimits`, атом радикала содержит поле ограничителя, чтобы указать, какой знак радикала был использован, Acc-атом содержит коды семейства и символа для акцента.

Когда вы в математической моде говорите `\hbox{...}`, в текущий математический список помещается атом Ord, ядром которого является h-бокс. Аналогично, `\vcenter{...}` производит Vcent-атом, ядро которого есть бокс. Но в большинстве случаев ядрами атомов будут либо символы, либо математические списки.

Вы можете поэкспериментировать с `\showlists`, чтобы понять внутреннее представление таких штук, как дроби и математический выбор.

Глава 26 содержит полное описание того, как создаются математические списки. Как только оканчивается математическая мода (т.е., встречается закрывающийся `$`), TEX разбирает текущий математический список и преобразовывает его в горизонтальный. Правила этого преобразования перечислены в приложении G. Вы можете увидеть “до” и “после” представление такого математического набора, оканчивая формулу `\showlists$\showlists`; первый `\showlists` будет показывать математический список, а второй — (возможно, сложный) горизонтальный список, который создан из него.

Время обучения коротко.
Несколько минут дают общее представление,
а печатание пары страниц статьи обычно
устраняет большинство недоразумений.

The learning time is short.
A few minutes gives the general flavor,
and typing a page or two of a paper
generally uncovers most of the misconceptions.

— KERNIGHAN and CHERRY, *A System for Typesetting Mathematics* (1975)

За несколько часов (самое большее — дней)
наборщика, не имеющий печатного или
математического опыта, можно научить
вводу даже самых сложных уравнений.

Within a few hours (a few days at most)
a typist with no math or typesetting
experience can be taught to input
even the most complex equations.

—PETER J.BOEHM, *Software and HardwareConsiderations for a Technical Typesetting System* (1976)

18

Математика: тонкости набора

Мы обсудили массу возможностей для создания математических формул, но хорошему наборщику этого мало. Когда вы введете приблизительно дюжину формул, используя идеи глав 16 и 17, вы обнаружите, что можете легко представить, каков будет окончательный вид математического выражения, которое вы вводите. А поскольку вы достигли этого уровня, осталось узнать только еще немножко — и у вас будут получаться такие прекрасные формулы, каких никто в мире даже и не видел. Примененные со вкусом приемы TEXники добавят профессиональную отделку, которая удивительно влияет на внешний вид и читабельность книг и документов. Эта глава говорит о таких приемах и, упоминая некоторые аспекты математики, которые не вписались в главы 16 и 17, заполняет некоторые пробелы в ваших знаниях.

1. Пунктуация. Когда за формулой следует точка, запятая, точка с запятой, двоеточие, вопросительный знак, восклицательный знак и т.п., ставьте знаки пунктуации *после* `$`, если формула в тексте, и *перед* `$$`, если формула выделена. Например:

```
Если $x<0$, мы показали, что $$y=f(x).$$
```

Правила TEX'а для распределения пробелов внутри абзацев работают лучше, когда знаки пунктуации не рассматриваются как часть формул.

Аналогично, не вздумайте вводить что-то вроде

```
для $x = a, b$, или $c$.
```

Надо вводить

```
для $x = a$, $b$, или $c$.
```

(Еще лучше использовать связку : "`или~$c$`".) Дело в том, что TEX будет печатать выражение `$x = a, b$` как одну формулу, поэтому он поставит "тонкий пробел" между запятой и b. Этот пробел не будет таким же, как пробел, который TEX ставит после запятой *после* b, поскольку пробелы между словами всегда больше тонких пробелов. Такие неравномерные пробелы неважно выглядят, но если вы все напечатаете правильно, пробелы будут выглядеть красиво.

Другая причина, чтобы не печатать `$x = a, b$`, заключается в том, что это сдерживает возможности разбиения строк в абзаце. TEX никогда не разорвет пробел между запятой и `b`, поскольку обычно формулы после запятой не разрываются. Например, в равенстве `$x = f(a, b)$` мы, конечно, не хотим помещать "$x = f(a,$" на одной строке, а "$b)$" — на другой.

Таким образом, когда вы вводите формулы в тексте, сохраняйте правильное выделение математики: не используйте такие операторы, как $-$ и $=$ за знаками `$` и сохраняйте запятые внутри формулы, если они действительно часть формулы. Но если запятая, точка или другие знаки пунктуации принадлежат скорее лингвистическому предложению, чем формуле, оставьте их вне знаков `$`.

▸ **Упражнение 18.1**
Напечатайте формулу: $R(n,t) = O(t^{n/2})$, при $t \to 0^+$.

Некоторые стили печати вставляют вокруг формул маленькие дополнительные пробелы, чтобы отделить их от текста. Например, когда печатается экземпляр на обычной пишущей машинке, которая не содержит курсивных букв, опытные технические машинистки традиционно ставят дополнительные пробелы перед и после каждой формулы, поскольку это обеспечивает визуальное выделение. Может быть полезно рассматривать каждый `$` как символ, который добавляет маленький пробел к напечатанному результату; тогда правила об исключении знаков пунктуации предложения из формул легче запомнить.

TEX фактически вставляет дополнительный пробел перед и после формулы. Величина такого пробела называется `\mathsurround`, что представляет собой параметр типа ⟨размер⟩. Например, если вы укажете `\mathsurround=1pt`, то каждая формула окажется на 2 pt шире (по 1 pt с каждой стороны):

Для $x = a$, b или c. (`\mathsurround=1pt`)
Для $x = a$, b или c. (`\mathsurround=0pt`)

Если формула встретилась в начале или в конце строки, то дополнительные пробелы, соответственно, слева или справа исчезают. То значение `\mathsurround`, которое действует, когда TEX читает закрывающий `$` формулы, используется как слева, так и справа этой формулы. Начальный TEX принимает `\mathsurround=0pt`, так что, если только вы не используете какой-нибудь другой формат или сами не изменили значение `\mathsurround`, вы не увидите никаких дополнительных пробелов.

2. Некурсивные буквы в формулах. Именами алгебраических переменных обычно являются курсивные или греческие буквы, но общепринятые математические функции типа “log” всегда печатаются прямым шрифтом. Чтобы иметь дело с такими конструкциями, лучше всего использовать следующие 32 команды (которые все определены в формате начального TEX’а в приложении B):

```
\arccos  \cos   \csc   \exp   \ker     \limsup  \min  \sinh
\arcsin  \cosh  \deg   \gcd   \lg      \ln      \Pr   \sup
\arctan  \cot   \det   \hom   \lim     \log     \sec  \tan
\arg     \coth  \dim   \inf   \liminf  \max     \sin  \tanh
```

Эти команды приводят к романскому шрифту с соответствующими пробелами:

Вход	Выход
`$\sin2\theta=2\sin\theta\cos\theta$`	$\sin 2\theta = 2\sin\theta\cos\theta$
`$O(n\log n\log\log n)$`	$O(n \log n \log\log n)$
`$\Pr(X>x)=\exp(-x/\mu)$`	$\Pr(X > x) = \exp(-x/\mu)$
`$$\max_{1\le n\le m}\log_2P_n$$`	$\max\limits_{1\le n\le m} \log_2 P_n$

```
$$\lim_{x\to0}{\sin x\over x}=1$$
```
$$\lim_{x\to0}{\sin x\over x}=1$$

Последние две формулы, которые являются выделенными, показывают, что TEX трактует некоторые специальные команды как "большие операторы" с пределами (типа $\sum$). Нижний индекс в `\max` трактуется не так, как нижний индекс в `\log`. Верхние и нижние индексы становятся пределами, когда они в выделенном стиле присоединены к `\det`, `\gcd`, `\inf`, `\lim`, `\liminf`, `\limsup`, `\max`, `\min`, `\Pr` и `\sup`.

▸ **Упражнение 18.2**
Выразите следующее выделенное равенство на языке начального TEX'а, используя `\nu` для ν:

$$p_1(n) = \lim_{m\to\infty} \sum_{\nu=0}^{\infty} \bigl(1 - \cos^{2m}(\nu!^n \pi/n)\bigr).$$

Если вам нужен прямой шрифт для некоторых математических функций или операторов, которые не включены в список 32 функций начального TEX'а, то легко определить новую команду по аналогии с определениями приложения B. Или если вам нужен прямой шрифт только для однократного использования, легче получить желаемое, включив `\rm`, как это показано дальше:

`$\sqrt{{\rm Var}(X)}$`	$\sqrt{\rm Var}(X)$
`$x_{\rm max}-x_{\rm min}$`	$x_{\rm max} - x_{\rm min}$
`${\rm LL}(k)\Rightarrow{\rm LR}(k)$`	${\rm LL}(k) \Rightarrow {\rm LR}(k)$
`$\exp(x+{\rm constant})$`	$\exp(x + {\rm constant})$
`$x^3+{\rm другие\ члены}$`	$x^3 + {\rm другие\ члены}$

Отметим использование "`\␣`" в последнем случае; без этого результатом было бы $x^3 + $ другиечлены, потому что в математической моде обычные пробелы игнорируются.

Для того, чтобы получить в формулах прямой шрифт, вместо `\rm` можно использовать `\hbox`. Например, четыре последние из приведенных пяти формул можно генерировать так:

`$\sqrt{\hbox{Var}(X)}$`	$\sqrt{\hbox{Var}(X)}$
`$\hbox{LL}(k)\Rightarrow\hbox{LR}(k)$`	$\hbox{LL}(k) \Rightarrow \hbox{LR}(k)$
`$\exp(x+\hbox{constant})$`	$\exp(x + \hbox{constant})$
`$x^3+\hbox{другие члены}$`	$x^3 + \hbox{другие члены}$

В этом случае `\␣` необязательно, потому что материал `\hbox` обрабатывается в горизонтальной моде, где пробелы являются значимыми. Но такое применение `\hbox` имеет два недостатка. (1) Содержимое бокса будет напечатано в одном и том же размере, независимо от того, встречается бокс в нижнем индексе или нет, например, `$x_{\hbox{max}}$` дает $x_{\hbox{max}}$. (2) В `\hbox` будет использоваться "текущий шрифт," а он не обязательно будет прямым. Например, если вы печатаете утверждение некоторой теоремы в наклонном шрифте и если эта теорема упоминает `$\sqrt{\hbox{Var}(X)}$`, вы получите неожиданный результат $\sqrt{\hbox{Var}(X)}$.

Для уверенности в том, что `\hbox` использует прямой шрифт, надо задать `\rm`, т.е., `$\sqrt{\hbox{\rm Var}(X)}$`; и тогда `\hbox` излишен. Однако позже мы увидим, что `\hbox` может быть очень полезным в выделенных формулах.

▸ **Упражнение 18.3**
Когда выделенная формула `$$\lim_{n\to\infty}x_n {\rm\ существует} \iff \limsup_{n\to\infty}x_n = \liminf_{n\to\infty}x_n.$$` печатается стандартными макрокомандами начального TEX'а, получается

$$\lim_{n\to\infty} x_n \text{ существует} \iff \limsup_{n\to\infty} x_n = \liminf_{n\to\infty} x_n.$$

Но некоторые предпочитают другую запись. Объясните, как можно изменить определения `\limsup` и `\liminf`, чтобы выделенная формула была бы такой:

$$\lim_{n\to\infty} x_n \text{ существует} \iff \overline{\lim_{n\to\infty}}\, x_n = \underline{\lim_{n\to\infty}}\, x_n.$$

Слово mod в формулах обычно задается в прямом шрифте, но оно требует дополнительного внимания, поскольку используется двумя различными способами, которые требуют двух различных трактовок. Начальный TEX для этих двух случаев имеет две различные команды `\bmod` и `\pmod`: `\bmod` используется, когда mod является бинарной операцией (т.е. когда встречается между двумя величинами, как знак плюс), а `\pmod` — когда mod заключено в скобки в конце формулы. Например,

`$\gcd(m,n)=\gcd(n,m\bmod n)$`	$\gcd(m,n) = \gcd(n, m \bmod n)$
`$x\equiv y+1\pmod{m^2}$`	$x \equiv y+1 \pmod{m^2}$

`b` в `\bmod` обозначает "бинарный", а `p` в `\pmod` — "скобочный"("parenthesized"). Заметим, что `\pmod` вставляет свои собственные скобки. Величина, которая появляется после mod в скобках, должна быть заключена в фигурные скобки, если это не один символ.

▸ **Упражнение 18.4**
Что получит бедный B. L. User, когда введет `$x\equiv0 (\pmod y^n)$` ?

▸ **Упражнение 18.5**
Объясните, как получить $\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}\binom{n \bmod p}{k \bmod p} \pmod p$.

Тот же механизм, который работает в формулах для прямого шрифта, может быть с успехом использован, чтобы получить другие шрифты. Например, `\bf` дает жирный шрифт:

`$\bf a+b=\Phi_m$`	$\mathbf{a+b=\Phi_m}$

Заметим, что в этом примере выделилась жирным шрифтом не вся формула: + и = остались такими же, какими были. Начальный TEXработает так, что команды типа `\rm` и `\bf` действуют только на заглавные буквы от `A` до `Z`, строчные буквы от `a` до `z`, цифры от `0` до `9`, заглавные греческие буквы от `\Gamma` до `\Omega`, и математические акценты типа `\hat` и `\tilde`. Между прочим, в этих примерах не были использованы фигурные скобки, потому что знак `$` действует как символ группирования: `\bf` изменяет текущий шрифт, но это изменение локально, поэтому не действует на шрифт, который был текущим вне формулы.

В качестве жирного шрифта в начальном TEX'е лучше использовать "прямой жирный", а не "жирный курсив", поскольку последний нужен реже. Однако, если надо, TEX без труда может установить для жирного шрифта курсив (см. упражнение 17.20). Более широкие наборы математических шрифтов включают также шрифты script (рукописный), Fraktur и "blackboard bold". Начальный TEX не имеет их, но другие форматы, например, $\mathcal{AMS}$-TEX, имеют.

Кроме `\rm` и `\bf`, в формулах можно использовать `\cal`, чтобы получать заглавные буквы в "каллиграфическом" шрифте. Например, `$\cal A$` дает $\mathcal{A}$, а `$\cal Z$` — $\mathcal{Z}$. Но осторожно, это работает только с буквами от `A` до `Z`: если вы примените `\cal` к строчным или греческим буквам, то получите роковой результат.

Команда `\mit`, обозначающая "математический курсив" заглавные греческие буквы ($\Gamma, \ldots, \Omega$) преобразует в ($\mathit{\Gamma}, \mathit{\Delta}, \mathit{\Theta}, \mathit{\Lambda}, \mathit{\Xi}, \mathit{\Pi}, \mathit{\Sigma}, \mathit{\Upsilon}, \mathit{\Phi}, \mathit{\Psi}, \mathit{\Omega}$). Когда действует `\mit`, обычные буквы не меняются, они, как обычно, заданы курсивом, потому что берутся из математического курсивного шрифта. Наоборот, на заглавные греческие буквы и на математические акценты не действует `\rm`, потому что они обычно берутся из прямого шрифта. Математические акценты не должны использоваться, когда выбирается семейство `\mit`, поскольку математический курсивный шрифт не содержит акцентов.

▸ **Упражнение 18.6**
Введите формулу $\bar{\mathbf{x}}^T\mathbf{M}\mathbf{x} = 0 \iff \mathbf{x} = \mathbf{0}$, используя как можно меньше нажатий клавиш. (Первый 0 — простого прямого шрифта, второй — жирного).

▸ **Упражнение 18.7**
Напишите, как ввести $S \subseteq \Sigma \iff S \in \mathcal{S}$.

Начальный TEX также позволяет вводить `\it`, `\sl` или `\tt`, если в математических формулах встречается текст, напечатанный курсивным, наклонным шрифтом или шрифтом пишущей машинки. Но эти шрифты имеются только в текстовом размере, поэтому не пытайтесь использовать их в индексах.

Если вы внимательны, то, вероятно, удивляетесь, почему предусмотрены как `\mit`, так и `\it`. Ответ в том, что `\mit` — это "математический курсив" (который обычно лучше для формул), а `\it` — "текстовый курсив" (который обычно лучше для сплошного текста).

```
$Это\ математический\ курсив.$      Это математический курсив.
{\it Это текстовый курсив.}         Это текстовый курсив.
```

Математические курсивные буквы немного шире, к тому же иначе расставляются пробелы. То, что отлично выглядит в математических формулах, терпит фиаско, когда вы пытаетесь напечатать курсивное слово *different*, используя математическую моду. (`$different$`). Широкое f обычно желательно в формулах и нежелательно в тексте. Поэтому умные наборщики используют `\it` в математических формулах, которые содержат курсивные слова. Такие случаи почти не встречаются в классической математике, но обычны для компьютерных программ, поскольку в них часто используют многобуквенные "идентификаторы":

```
$\it last:=first$                   last := first
```

```
$\it x\_coord(point\_2)$
```
$x\_coord(point\_2)$

Первый из этих примеров показывает, что когда курсив встречается в математической формуле, TEX распознает лигатуру *fi*. Другой пример иллюстрирует использование короткого подчеркивания для разделения имен идентификаторов. Когда автор готовил это руководство, то для ссылок на стиль *SS* он использовал `$\it SS$`, поскольку '`$SS$`' делает буквы S стоящими отдельно: SS.

▸ **Упражнение 18.8**
Какие команды начального TEX'а выдадут следующую выделенную формулу?

$$available + \sum_{i=1}^{n} \max\bigl(full(i), reserved(i)\bigr) = capacity.$$

▸ **Упражнение 18.9**
Как бы вы приступили к следующей компьютерной программе, используя макрокоманды начального TEX'а?

```
for j := 2 step 1 until n do
  begin accum := A[j]; k := j - 1; A[0] := accum;
  while A[k] > accum do
    begin A[k+1] := A[k]; k := k - 1;
    end;
  A[k+1] := accum;
  end.
```

3. Пробелы между формулами. Выделенные выражения часто содержат не просто одну формулу. Например, уравнение часто сопровождается граничными условиями:

$$F_n = F_{n-1} + F_{n-2}, \qquad n \ge 2.$$

В таких случаях вам надо указать TEX'у, какой пробел оставлять после запятой, потому что обычные соглашения TEX'а, если вы не приняли специальных предосторожностей, сдвинули бы все вместе:

$$F_n = F_{n-1} + F_{n-2}, n \ge 2.$$

Традиционная технология металлического набора привела к некоторым укоренившимся стандартам для ситуаций, подобных этой, основанных на том, что наборщики называют "квадратом" пространства. Поскольку оказывается, что эти стандарты на практике хорошо работают, с помощью TEX'а легко продолжить эту традицию. Если ввести `\quad` в формате начального TEX'а, то получится полиграфический квадрат в горизонтальном направлении. Аналогично, `\qquad` дает двойной квадрат (вдвое больший). Это нормальное распределение пробелов для ситуации в примере F_n, приведенном выше. Итак, в этом случае рекомендуется процедура:

```
$$ F_n = F_{n-1} + F_{n-2}, \qquad  n \ge 2. $$
```

Возможно стоит повторить, что в математической моде TEX игнорирует все пробелы (за исключением, конечно, пробела после `\qquad`, который нужен,

чтобы отличить `\qquad n` от `\qquadn`). Поэтому, чтобы получить тот же самый результат, можно выкинуть все пробелы, кроме одного:

```
$$F_n=F_{n-1}+F_{n-2},\qquad n\ge2.$$
```

Всякий раз, когда вы хотите использовать распределение пробелов, которое отличается от обычных соглашений, вы должны указать пробелы явно, используя такие команды, как `\quad` и `\qquad`.

Квадрат обычно бывает чистым квадратным кусочком 1 em шириной и 1 em высотой, приблизительно размера заглавной буквы M, как это объяснялось в главе 10. Эта традиция не полностью сохранена: команда `\quad` начального TEX'а — это просто сокращение для `\hskip 1em\relax`, поэтому квадрат TEX'а имеет ширину, но не имеет высоты.

Можно использовать `\quad` как в формулах, так и в тексте. Например, глава 14 иллюстрирует, как `\quad` применяется при печати стихов. Когда `\quad` появляется в формуле, он обозначает один em текущего текстового шрифта, независимо от текущего математического размера, стиля или семейства. Так, например, `\quad` имеет ту же ширину как в индексах, так и на основной линии формулы.

Иногда невнимательный автор может поставить в тексте абзаца две формулы подряд. Например, можно встретить такое предложение:

Числа Фибоначчи удовлетворяют условию $F_n = F_{n-1} + F_{n-2},\ \ n \ge 2.$

Каждый, кто изучал математический стиль, знает, что формулы должны отделяться словами, а не только запятой. Автору этого предложения следовало бы как минимум сказать "для $n \ge 2$", а не просто "$n \ge 2$". Но, увы, такой грех банален, и многие выдающиеся математики безнадежно увлекаются скученными формулами. Если не позволено изменить их стиль написания формул, можно как минимум вставить дополнительные пробелы там, где они не удосужились вставить подходящее слово. В таких случаях обычно хорошо работает дополнительный пробел между словами. Например, указанное выше предложение было подготовлено так:

```
... $F_n=F_{n-1}+F_{n-2}$, \ $n\ge2$.}$$
```

Здесь `\␣` дает визуальное разделение, которое частично компенсирует плохой стиль.

▸ **Упражнение 18.10**

Представьте следующий абзац в TEXовской форме, внимательно отрабатывая пунктуацию и пробелы. Вставьте также связки для того, чтобы избе-

жать плохого разбиения строк.

> Пусть H — Гильбертово пространство, C — замкнутое ограниченное выпуклое подмножество пространства H, T — нерасширенное отображение C. Предположим, что при $n \to \infty$, $a_{n,k} \to 0$ для каждого k, и $\gamma_n = \sum_{k=0}^{\infty}(a_{n,k+1} - a_{n,k})^+ \to 0$. Тогда для каждого x в C $A_n x = \sum_{k=0}^{\infty} a_{n,k} T^k x$ слабо сходится к фиксированной точке T.

4. Пробелы внутри формул. Глава 16 говорит, что TEX в математических формулах автоматически ставит такие пробелы и что они выглядят правильно, и это почти правда. Но иногда надо оказать TEX'у некоторую помощь. Число возможных математических формул безгранично, а правила TEX'а по расстановке пробелов довольно просты, поэтому, естественно, должны возникать исключения. Конечно, желательно для такой цели иметь хорошие единицы пробелов вместо больших интервалов, которые получаются из `\␣`, `\quad` и `\qquad`.

Основные виды промежутков, которые TEX вставляет в формулы, называются *тонкими пробелами*, *средними пробелами*, и *толстыми пробелами*.Для того, чтобы почувствовать эти единицы, давайте снова посмотрим на пример F_n. Толстые пробелы встречаются перед и после знака =, а также перед и после $\geq$. Средние пробелы — перед и после знака +. Тонкие пробелы несколько меньше, но заметны: именно тонкие пробелы создают различия между loglog и log log. Обычно расстояние между словами абзаца приблизительно равно двум тонким пробелам.

TEX автоматически вставляет в формулы тонкие, средние и толстые пробелы, но можно добавлять и свои собственные пробелы, используя для этого команды

`\,`	тонкий пробел (обычно 1/6 квадрата);
`\>`	средний пробел (обычно 2/9 квадрата);
`\;`	толстый пробел (обычно 5/18 квадрата);
`\!`	отрицательный тонкий пробел (обычно $-1/6$ квадрата).

В большинстве случаев при подготовке рукописи можно полагаться на пробелы TEX'а, а вставлять и удалять пробелы этими четырьмя командами только в редких случаях, когда уже видно, что из этого получается.

Только что мы видели, что пробел `\quad` не изменяется с изменением стиля формулы, а также не зависит от семейств используемых математических шрифтов. Но тонкие, средние и толстые пробелы увеличиваются и уменьшаются вместе с увеличением и уменьшением размера шрифта. Это происходит потому, что они определены через математический клей (muglue), специальный вид клея, предназначенный для математических пробелов. Вы указываете математический клей так же, как если бы это был обычный клей, за тем исключением, что единицы задаются в `mu` (математические единицы), а не в `pt`,`cm` или в чем-

нибудь еще. Например, приложение B содержит определения

```
\thinmuskip = 3mu
\medmuskip = 4mu plus 2mu minus 4mu
\thickmuskip = 5mu plus 5mu
```

Они определяют тонкие, средние и толстые пробелы, которые TEX вставляет в формулы. В соответствии с этими спецификациями тонкие пробелы начального TEX'а не могут ни растягиваться, ни сжиматься, средние пробелы могут немного растягиваться и сжиматься до нуля, толстые пробелы могут сильно растягиваться, но никогда не сжимаются.

Один em равен 18 mu, где em берется из семейства 2 (семейство математических символов). Другими словами, `\textfont 2` определяет значение em для `mu` выделенного и текстового стилей, `\scriptfont 2` определяет em для индексного размера, а `\scriptscriptfont 2` определяет его для размера повторного индекса.

Можно вставить математический клей в любую формулу, просто задавая '`\mskip`⟨математический клей⟩'. Например, '`\mskip 9mu plus 2mu`' вставляет промежуток, равный половине em текущего размера с некоторой растяжимостью. Приложение B определяет `\,` как сокращение для `\mskip\thinmuskip`. Аналогично, когда нет растяжения и сжатия, можно использовать команду `\mkern`: `\mkern18mu` дает один em горизонтального пробела в текущем размере. TEX настаивает на том, чтобы `\mskip` и `\mkern` использовались только с `mu`. Наоборот, `\hskip` и `\kern` (которые также разрешены в формулах) никогда не должны задаваться в единицах `mu`.

Формулы, содержащие дифференциалы, выглядят лучше, когда перед dx, dy или d что-то вставлен дополнительный тонкий пробел, но TEX автоматически этого не делает. Поэтому опытный наборщик всегда будет помнить, что в таких случаях надо вставлять '`\,`':

Input	*Output*
`$\int_0^\infty f(x)\,dx$`	$\int_0^\infty f(x)\,dx$
`$y\,dx-x\,dy$`	$y\,dx-x\,dy$
`$dx\,dy=r\,dr\,d\theta$`	$dx\,dy=r\,dr\,d\theta$
`$x\,dy/dx$`	$x\,dy/dx$

Заметим, что в последнем примере желательно отсутствие '`\,`' после '`/`'. Аналогично, нет необходимости в '`\,`' в таких случаях, как

```
$$\int_1^x{dt\over t}$$
```

$$\int_1^x \frac{dt}{t}$$

поскольку dt появляется в одиночестве в числителе дроби, а это визуально отделяет его от остальной формулы.

▸ **Упражнение 18.11**

Объясните, как напечатать следующую выделенную формулу:

$$\int_0^\infty \frac{t-ib}{t^2+b^2} e^{iat}\,dt = e^{ab}E_1(ab), \qquad a,b>0.$$

Когда в формулах появляются физические единицы, они должны печататься латинским шрифтом и отделяться от предшествующего материала тонкими пробелами:

`$55\rm\,mi/hr$`	$55\,\mathrm{mi/hr}$
`$g=9.8\rm\,m/sec^2$`	$g = 9.8\,\mathrm{m/sec}^2$
`$\rm1\,ml=1.000028\,cc$`	$1\,\mathrm{ml} = 1.000028\,\mathrm{cc}$

▸ **Упражнение 18.12**

Напечатайте следующее выделенное равенство, предполагая, что `\hbar` генерирует $\hbar$:

$$\hbar = 1.0545 \times 10^{-27}\,\mathrm{erg\,sec}.$$

Тонкие пробелы должны также вставляться после восклицательного знака (который в формулах обозначает операцию “факториал”), если за ним следует буква, цифра или открывающий ограничитель:

`$(2n)!/\bigl(n!\,(n+1)!\bigr)$`	$(2n)!/\bigl(n!\,(n+1)!\bigr)$
`$${52!\over13!\,13!\,26!}$$`	$$\frac{52!}{13!\,13!\,26!}$$

Кроме того, вы иногда будете сталкиваться с формулами, в которых символы сдвинуты слишком тесно или слишком много пустого места из-за некоторого неудачного сочетания форм. Обычно невозможно предвидеть такие оптические эффекты, пока вы не увидите первые гранки того, что печатали. Используйте свой здравый смысл для того, чтобы добавить завершающие штрихи, которые обеспечивают дополнительную красоту, прозрачность и тонкость. Примененные со вкусом `\,` или `\!` раздвинут или сблизят элементы так, что читатель не будет отвлекаться от математического смысла формулы. Часто кандидатами на такую тонкую настройку являются квадратные корни и многократные интегралы. Приведем несколько примеров таких ситуаций:

`$\sqrt2\,x$`	$\sqrt2\,x$
`$\sqrt{\,\log x}$`	$\sqrt{\log x}$
`$0\bigl(1/\sqrt n\,\bigr)$`	$0\bigl(1/\sqrt n\,\bigr)$
`$[\,0,1)$`	$[\,0,1)$
`$\log n\,(\log\log n)^2$`	$\log n\,(\log\log n)^2$
`$x^2\!/2$`	$x^2\!/2$
`$n/\!\log n$`	$n/\!\log n$
`$\Gamma_{\!2}+\Delta^{\!2}$`	$\Gamma_{\!2}+\Delta^{\!2}$

```
$R_i{}^j{}_{\!kl}$                $R_i{}^j{}_{\!kl}$
$\int_0^x\!\int_0^y dF(u,v)$      $\int_0^x\!\int_0^y dF(u,v)$
$$\int\!\!\!\int_D dx\,dy$$       $$\int\!\!\!\int_D dx\,dy$$
```

В каждой из этих формул пропуск '`\,`' или '`\!`' привел бы к менее удовлетворительным результатам.

Большинство случаев, в которых желательна коррекция тонкими пробелами, возникает из-за случайных совпадений. Например, верхний индекс в `$x^2/2$` оставляет дырку перед / ($x^2/2$), а отрицательный тонкий пробел помогает заполнить эту дырку. Положительный тонкий пробел в `$\sqrt{\,\log x}$` компенсирует тот факт, что $\log x$ начинается с высокой ненаклонной буквы; и так далее. Но два примера содержат коррекции, которые оказались необходимыми потому, что TEX плохо знает математику. (1) В формуле `$\log n(\log\log n)^2$`, TEX не вставляет тонкий пробел перед левой скобкой, потому что бывают похожие формулы типа `$\log n(x)$`, в которых такой пробел нежелателен. (2) В формуле `$n/\log n$` TEX автоматически вставляет нежелательный тонкий пробел перед `\log`, поскольку наклонная черта трактуется как обычный символ и поскольку между обычным символом и оператором типа `\log` обычно желателен тонкий пробел.

На самом деле правила TEX'а по расстановке пробелов в формулах совершенно просты. Формула преобразуется в математический список, как это описано в конце главы 17. Этот список состоит главным образом из "атомов" восьми основных типов: Ord (ординарный или обычный), Op (большой оператор), Bin (бинарная операция), Rel (отношение), Open (открывающий), Close (закрывающий), Punct (пунктуация) и Inner (ограниченная подформула). Другие виды атомов, которые возникают из таких команд, как `\overline`, `\mathaccent`, `\vcenter` и т.п., все трактуются как Ord, а дроби считаются типа Inner. Для определения пробелов между парами соседних атомов используется следующая таблица:

		Правый атом							
		Ord	Op	Bin	Rel	Open	Close	Punct	Inner
	Ord	0	1	(2)	(3)	0	0	0	(1)
	Op	1	1	*	(3)	0	0	0	(1)
	Bin	(2)	(2)	*	*	(2)	*	*	(2)
Левый	Rel	(3)	(3)	*	0	(3)	0	0	(3)
атом	Open	0	0	*	0	0	0	0	0
	Close	0	1	(2)	(3)	0	0	0	(1)
	Punct	(1)	(1)	*	(1)	(1)	(1)	(1)	(1)
	Inner	(1)	1	(2)	(3)	(1)	0	(1)	(1)

Здесь 0, 1, 2 и 3 обозначают, соответственно, отсутствие пробела, тонкий пробел, средний пробел и толстый пробел. Элемент таблицы заключен в скобки, если пробел вставляется только в выделенном и текстовом стилях, а не в стилях индекса или повторного индекса. Например, в ряде Rel колонки Rel часто встречается (3). Это означает, что обычно перед и после символов отношения типа = вставляется

толстый пробел, но в индексах он не вставляется. Некоторые элементы таблицы равны `*`. Такие случаи никогда не возникают, поскольку атомам Bin должны предшествовать, а также и следовать за ними, атомы, совместимые с природой бинарных операций. Приложение G содержит точное описание того, как математические списки преобразуются в горизонтальные. Это преобразование делается, когда TEX собирается выйти из математической моды, тогда и вставляются междуатомные пробелы.

Например, спецификация выделенной формулы

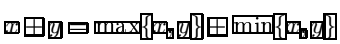
```
$$x+y=\max\{x,y\}+\min\{x,y\}$$
```

будет преобразована в последовательность атомов

соответственно, типов Ord, Bin, Ord, Rel, Op, Open, Ord, Punct, Ord, Close, Bin, Op, Open, Ord, Punct, Ord и Close. Вставка пробелов в соответствии с таблицей дает

```
Ord \> Bin \> Ord \; Rel \; Op Open Ord Punct \, Ord Close \>
                                    Bin \> Op Open Ord Punct \, Ord Close
```

и результирующая формула будет



т.е.,

$$x + y = \max\{x, y\} + \min\{x, y\} \quad .$$

Этот пример не содержит верхних и нижних индексов. Индексы просто присоединяются к атомам без изменения их типа.

▸ **Упражнение 18.13**
Используя таблицу определите, какие пробелы вставит TEX между атомами формулы `$f(x,y)<x^2+y^2$`.

Макрокоманды начального TEX'а `\bigl`, `\bigr`, `\bigm` и `\big` производят идентичные ограничители. Единственное отличие между ними в том, что они могут привести к различной расстановке пробелов, поскольку преобразуют ограничители в атомы разных типов: `\bigl` производит атом Open, `\bigr` — Close, `\bigm` — Rel, а `\big` — Ord. Когда подформула появляется между `\left` и `\right`, она обрабатывается и помещается в атом Inner. Поэтому возможно, что подформула, заключенная между `\left` и `\right`, будет окружена более широкими пробелами, чем если бы она была заключена между `\bigl` и `\bigr`. Например, Ord, за которым следует Inner (из `\left`), получает тонкий пробел, а если за Ord следует Open (из `\bigl`), то пробела не будет. Правила из главы 17 подразумевают, что конструкция `\mathinner{\bigl({`⟨подформула⟩`}\bigr)}` внутри любой формулы дает результат, эквивалентный результату `\left(`⟨подформула⟩`\right)`, за тем исключением, что ограничителям придется быть размера `\big` независимо от высоты и глубины подформулы.

Правила расстановки пробелов TEX'а иногда дают сбой, когда в формулах встречаются | и \|, потому что | и || трактуются, как обычные символы (Ord), а не как ограничители. Например, рассмотрим формулы

`$|-x|=|+x|$`	$\|-x\|=\|+x\|$
`$\left|-x\right|=\left|+x\right|$`	$\left\|-x\right\|=\left\|+x\right\|$
`$\lfloor-x\rfloor=-\lceil+x\rceil$`	$\lfloor-x\rfloor=-\lceil+x\rceil$

В первом случае пробелы неправильны, поскольку TEX думает, что знак плюс вычисляет сумму | и x. Применение `\left` и `\right` во втором случае наставляет TEX на правильный путь. Третий пример показывает, что с другими ограничителями такой коррекции не требуется, поскольку TEX различает, являются они открывающими или закрывающими.

▸ **Упражнение 18.14**
Некоторые своенравные математики используют квадратные скобки задом наперед, чтобы обозначить "открытый интервал". Объясните, как напечатать следующую эксцентричную формулу: $]-\infty,T[\times]-\infty,T[$.

▸ **Упражнение 18.15**
Изучите приложение G и определите, какие пробелы будут использованы в формуле `$x++1$`. Какой из знаков плюс будет рассматриваться как бинарная операция?

5. Многоточия ("три точки"). Математические документы выглядят намного лучше, если вы внимательно относитесь к тому, как в тексте и в формулах печатаются группы из трех точек. Хотя многоточие прекрасно выглядит, если печатается `...` на пишущей машинке, которая фиксирует пробелы, результат выглядит слишком тесным, когда используются шрифты: из `$x...y$` получается $x...y$, а такие узкие пробелы нежелательны, разве что в верхних и нижних индексах.

Многоточия можно задавать двумя различными видами точек, выше или ниже; в лучших традициях рекомендуются делать различие между этими двумя возможностями. Принято считать правильными формулы типа

$$x_1+\cdots+x_n \qquad \text{и} \qquad (x_1,\ldots,x_n),$$

а неправильными — типа

$$x_1+\ldots+x_n \qquad \text{и} \qquad (x_1,\cdots,x_n).$$

Формат начального TEX'а в приложении B позволяет элементарно решить проблему "трех точек", и любой может позавидовать тем прекрасным формулам, которые вы будете получать. Просто вводите `\ldots`, когда надо получить три нижних точки ($\ldots$), и `\cdots`, когда нужны вертикально центрированные точки ($\cdots$).

Лучше использовать `\cdots` между знаками $+$, $-$ и $\times$, а также между знаками $=$, $\le$, $\subset$ или другими аналогичными отношениями. Нижние

точки используются между запятыми, и когда что-то образует ряд безо всяких знаков между ними. Например:

`$x_1+\cdots+x_n$`	$x_1+\cdots+x_n$
`$x_1=\cdots=x_n=0$`	$x_1=\cdots=x_n=0$
`$A_1\times\cdots\times A_n$`	$A_1\times\cdots\times A_n$
`$f(x_1,\ldots,x_n)$`	$f(x_1,\ldots,x_n)$
`$x_1x_2\ldots x_n$`	$x_1x_2\ldots x_n$
`$(1-x)(1-x^2)\ldots(1-x^n)$`	$(1-x)(1-x^2)\ldots(1-x^n)$
`$n(n-1)\ldots(1)$`	$n(n-1)\ldots(1)$

▸ **Упражнение 18.16**
Напечатайте формулы $x_1+x_1x_2+\cdots+x_1x_2\ldots x_n$ и $(x_1,\ldots,x_n)\cdot(y_1,\ldots,y_n)=x_1y_1+\cdots+x_ny_n$. [*Намек.* Одиночная поднятая точка вызывается `\cdot`.]

Но есть важный специальный случай, в котором `\ldots` и `\cdots` не дают правильных пробелов, а именно, когда они появляются в самом конце формулы или непосредственно перед закрывающим ограничителем типа ‘)’. В таких ситуациях необходим дополнительный тонкий пробел. Например, рассмотрим такие предложения:

> Докажите, что $(1-x)^{-1}=1+x+x^2+\cdots$.
> Очевидно, что $a_i<b_i$ для $i=1,2,\ldots,n$.
> Коэффициенты c_0, c_1, …, c_n положительны.

Чтобы получить первое предложение, автор ввел

```
Докажите, что $(1-x)^{-1}=1+x+x^2+\cdots\,$.
```

Без “`\,`” точка получилась бы слишком близкой к `\cdots`. Аналогично, второе предложение было напечатано так:

```
Очевидно, что $a_i<b_i$ для $i=1$,~2, $\ldots\,$,~$n$.
```

Отметим использование связки, которая предотвращает плохие разбиения, как это объяснялось в главе 14. Такие многоточия часто встречаются в некоторых математических работах, поэтому начальный TEX позволяет в тексте абзаца просто сказать `\dots`, в качестве сокращения для `$\ldots\,$`. Так что третье предложение может быть введено так:

```
Коэффициенты $c_0$,~$c_1$, \dots,~$c_n$ положительны.
```

▸ **Упражнение 18.17**
B. C. Dull пытался сократить работу, введя второй пример таким образом:

```
Очевидно, что $a_i<b_i$ for~$i=1, 2, \ldots, n$.
```

Что в этом плохого?

▸ **Упражнение 18.18**
Как, по вашему мнению, автор ввел сноску в главе 4 этой книги?

6. Разбиение строк. Когда в абзаце встречается формула, TEX может разбить ее между строками. Это неизбежное зло, такое же, как перенос слов. Хочется избежать его, если только альтернатива не хуже.

Формула будет разбиваться только после символов отношения типа $=$, $<$ или $\rightarrow$, или после символов бинарной операции типа $+$, $-$ или $\times$, когда отношения или бинарные операции находятся на "внешнем уровне" формулы (т.е., не заключены в `{...}` и не являются частью конструкции `\over`). Например, если вы вводите

```
$f(x,y) = x^2-y^2 = (x+y)(x-y)$
```

в середине абзаца, то есть шанс, что TEX разорвет строку либо после знака `=` (он предпочитает это), либо после `-`, `+` или `-` (в крайнем случае). Но ни в коем случае не будет разрыва после запятой — запятая, после которой желателен разрыв, не должна появляться между знаками `$`.

Если в этом примере вы не хотите разрешать никаких других разрывов, кроме как после знака `=`, то можете ввести

```
$f(x,y) = {x^2-y^2} = {(x+y)(x-y)}$,
```

поскольку дополнительные фигурные скобки "спаивают" подформулы, помещая их в неразрываемые боксы, в которых устанавливается клей натуральной ширины. Но нет необходимости заранее суетиться по поводу таких вещей, пока TEX на самом деле неудачно не разорвет формулу, поскольку вероятность этого довольно мала.

В формулах позволяется "разрывный знак умножения": если вы введете `$(x+y)\*(x-y)$`, TEX будет трактовать `\*` таким же образом, как он трактует `\-`, а именно, в этом месте будет разрешен разрыв строки с таким же штрафом, как при переносе слов. Однако, вместо вставки знака переноса TEX вставит знак $\times$ в текстовом размере.

Если вы хотите разрешить разрыв в некоторой точке на внешнем уровне формулы, то можете сказать `\allowbreak`. Например, если формула

```
$(x_1,\ldots,x_m,\allowbreak y_1,\ldots,y_n)$
```

появляется в тексте абзаца, TEX позволяет разорвать ее на два куска ($x_1, \ldots, x_m$, и $y_1, \ldots, y_n$).

Штраф за разрыв после атома Rel называется `\relpenalty`, а штраф за разрыв после атома Bin — `\binoppenalty`. Начальный TEX устанавливает `\relpenalty=500` и `\binoppenalty=700`. Можно изменять штраф за разрыв в каждом конкретном случае, вводя `\penalty`⟨число⟩ сразу после атома, о котором идет речь. Тогда число, которое вы указали, будет использоваться вместо обычного штрафа. Например, можно запретить разрыв в формуле $x = 0$, введя `$x=\nobreak0$`, поскольку `\nobreak` — это сокращение для `\penalty10000` .

▸ **Упражнение 18.19**
Есть ли различие между результатами `$x=\nobreak0$` и `${x=0}$`?

▸ **Упражнение 18.20**
Как можно запретить все разрывы в формулах, сделав только небольшие изменения макрокоманд начального TEX'а?

7. Фигурные скобки. Много различных замечаний возникает из-за символов { и }. Начальный TEX имеет несколько команд, которые помогают справиться с формулами, использующими такие штуки.

В простых ситуациях фигурные скобки используются, чтобы указать множество объектов: например, $\{a, b, c\}$ обозначает множество из трех объектов a, b и c. Нет ничего особенного во вводе таких формул, только не забывайте, что для фигурных скобок надо использовать `\{` и `\}`:

`$\{a,b,c\}$`	$\{a, b, c\}$
`$\{1,2,\ldots,n\}$`	$\{1, 2, \ldots, n\}$
`$\{\rm красный,белый,синий\}$`	$\{\text{красный}, \text{белый}, \text{синий}\}$

Несколько более сложный случай возникает, когда множество задается указанием общего элемента со следующим за ним особым условием. Например, $\{\, x \mid x > 5 \,\}$ обозначает множество всех объектов x, которые больше 5. В таких случаях надо использовать команду `\mid` для вертикальной черты, а внутри фигурных скобок вставить тонкий пробел:

`$\{\,x\mid x>5\,\}$`	$\{\, x \mid x > 5 \,\}$
`$\{\,x:x>5\,\}$`	$\{\, x : x > 5 \,\}$

(Некоторые авторы предпочитают вместо | использовать двоеточие, как показано во втором примере.) Чтобы получить более крупные ограничители, как в

$$\bigl\{\, \bigl(x, f(x)\bigr) \bigm| x \in D \,\bigr\}$$

их надо вызвать с помощью `\bigl`, `\bigm` и `\bigr`. Например, последняя формула была введена так:

```
\bigl\{\,\bigl(x,f(x)\bigr)\bigm|x\in D\,\bigr\}
```

а формулы, которые привлекают еще большие ограничители, должны использовать `\Big`, `\bigg` или даже `\Bigg`, как объяснялось в главе 17.

▸ **Упражнение 18.21**
Как бы вы ввели формулу $\bigl\{\, x^3 \bigm| h(x) \in \{-1, 0, +1\} \,\bigr\}$?

▸ **Упражнение 18.22**
Иногда условия, которые определяют множество, заданы длинным словесным описанием, а не формулой. Например, рассмотрим $\{\, p \mid p$ и $p+2$ простые числа $\}$. Здесь h-бокс выполняет такую работу:

```
$\{\,p\mid\hbox{$p$ и $p+2$ простые числа}\,\}$
```

но в абзаце длинная формула типа этой приносит много хлопот, поскольку h-бокс нельзя разрывать между строками и поскольку клей внутри `\hbox` не отличается от клея между словами на строке, которая его содержит. Объясните, как такая формула могла быть напечатана с разрешенными разрывами строки. [*Намек.* Перемещайтесь туда и обратно между математической и горизонтальной модами.]

Выделенные формулы часто используют фигурные скобки другого сорта, чтобы указать выбор между различными альтернативами, как в конструкции

$$|x| = \begin{cases} x, & \text{если } x \ge 0; \\ -x, & \text{иначе.} \end{cases}$$

Можно ввести это командой `\cases`:

```
$$|x|=\cases{x,&если $x\ge0$;\cr
             -x,&иначе.\cr}$$
```

Рассмотрим подробно этот пример и заметим, что он использует символ `&`, который, как мы говорили в главе 7, был зарезервирован для специальных целей. Здесь впервые в этом руководстве мы встретили пример того, как `&` служит таким специальным целям. Каждый из случаев имеет две части, а `&` отделяет эти части. Слева от `&` находится математическая формула, которая неявно заключена в `$...$`; справа от `&` — обычный текст, который *не* заключен в `$...$`. Например, на второй строке "`-x,`" будет вводиться в математической моде, а "`иначе`" — в горизонтальной. Пробелы после `&` игнорируются. Случаев может быть любое количество, но не меньше двух. За каждым случаем должен следовать `\cr`. Заметим, что конструкция `\cases` печатает свою собственную '{' без парной ей '}'.

▸ **Упражнение 18.23**
Напечатайте выделенную формулу $f(x) = \begin{cases} 1/3 & \text{если } 0 \le x \le 1; \\ 2/3 & \text{если } 3 \le x \le 4; \\ 0 & \text{в других случаях.} \end{cases}$

Можно вставлять `\noalign{`⟨вертикальный материал⟩`}` прямо после `\cr` внутри `\cases`, как объясняется в главе 22, потому что `\cases` — это частный случай общих конструкций выравнивания, рассматриваемых в этой главе. Например, команду `\noalign{\vskip2pt}` можно использовать, чтобы поставить маленький дополнительный пробел между двумя случаями.

Над или под отдельными частями выделенной формулы можно поставить горизонтальные фигурные скобки, если использовать команды `\overbrace` или `\underbrace`. Такие конструкции рассматриваются как большие операторы типа `\sum`, так что над или под ними можно ставить пределы, указывая верхние или нижние индексы, как это сделано в следующих примерах:

```
$$\overbrace{x+\cdots+x}^{k\rm\;times}$$
```
$$\overbrace{x+\cdots+x}^{k\ \text{times}}$$

```
$$\underbrace{x+y+z}_{>\,0}.$$
```
$$\underbrace{x+y+z}_{>0}.$$

8. Матрицы. Теперь мы подошли к интересному разделу. Математики во многих различных областях науки любят создавать прямоугольные наборы формул, которые устроены из строк и столбцов. Такие наборы называются *матрицами*. Начальный TEX имеет команду `\matrix`, с помощью которой удобно обращаться с большинством наиболее общих типов матриц.

Например, предположим, вы хотите задать выделенную формулу

$$A = \begin{pmatrix} x-\lambda & 1 & 0 \\ 0 & x-\lambda & 1 \\ 0 & 0 & x-\lambda \end{pmatrix}.$$

Все, что вам надо, это ввести

```
$$A=\left(\matrix{x-\lambda&1&0\cr
                  0&x-\lambda&1\cr
                  0&0&x-\lambda\cr}\right).$$
```

Это очень похоже на конструкцию `\cases`, которую мы видели ранее: за каждой строкой матрицы следует `\cr`, а между элементами каждой строки используется знак `&`. Заметим, однако, что вокруг матрицы надо ставить свои собственные ограничители `\left` и `\right`; Это отличает `\matrix` от `\cases`, которая автоматически вставляет большую {. Причина в том, что `\cases` всегда вставляет левую фигурную скобку, а различные конструкции матриц используют различные ограничители. С другой стороны, круглые скобки используются чаще других ограничителей, поэтому если вы хотите, чтобы начальный TEX вставил круглые скобки, можно написать `\pmatrix`. Тогда приведенный выше пример сокращается:

```
$$A=\pmatrix{x-\lambda&...&x-\lambda\cr}.$$
```

▸ **Упражнение 18.24**
Получите выделенную формулу

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} u & x \\ v & y \\ w & z \end{pmatrix}$$

используя `\lgroup` и `\rgroup`.

Отдельные элементы матриц обычно располагаются в центре столбцов. Каждый столбец сделан той ширины, которая необходима, чтобы соответствовать содержащимся в ней элементам, а между столбцами оставлен пробел величиной в 1 квадрат. Если вы хотите прижать элемент к правому краю столбца, напишите перед ним `\hfill`, а если к левому краю столбца, напишите после него `\hfill`.

Каждый элемент матрицы обрабатывается отдельно от других ее элементов и вводится как математическая формула в текстовом стиле. Так, если вы в одном элементе сказали `\rm`, это не действует на другие элементы. Не пытайтесь говорить `{\rm x&y}`.

Матрицы часто появляются в форме характерных элементов, которая использует многоточия для указания тех строк и столбцов, которые опущены. Такие матрицы можно напечатать, ставя многоточия в эти строки и/или столбцы. Начальный TEX вдобавок к `\ldots` предусматривает для таких конструкций `\vdots` (вертикальные точки) и `\ddots` (диагональные точки). Например, характерная матрица

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

легко задается так:

```
$$A=\pmatrix{a_{11}&a_{12}&\ldots&a_{1n}\cr
             a_{21}&a_{22}&\ldots&a_{2n}\cr
             \vdots&\vdots&\ddots&\vdots\cr
             a_{m1}&a_{m2}&\ldots&a_{mn}\cr}$$
```

▸ **Упражнение 18.25**
Как можно заставить TEX получить вектор-столбец $\begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}$?

Иногда матрица окаймлена сверху и слева формулами, которые задают метки строк и столбцов. Для такой ситуации начальный TEX имеет специальную макрокоманду `\bordermatrix`. Например, выделенная формула

$$M = \begin{array}{c} \\ C \\ I \\ C' \end{array} \begin{array}{c} \begin{array}{ccc} C & I & C' \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ b & 1-b & 0 \\ 0 & a & 1-a \end{pmatrix} \end{array}$$

получается, если вы вводите

```
$$M=\bordermatrix{&C&I&C'\cr
                  C&1&0&0\cr  I&b&1-b&0\cr  C'&0&a&1-a\cr}$$
```

Первый ряд задает верхние метки, которые появляются над левыми и правыми круглыми скобками, а первый столбец дает левые метки, которые печатаются слева непосредственно перед самой матрицей. Элемент первого столбца первой строки обычно пустой. Заметим, что как и `\bordermatrix`, `\pmatrix` вставляет свои собственные скобки.

Обычно нецелесообразно вставлять матрицы в текст абзаца, т.к. они настолько велики, что их лучше выделить. Но иногда может понадобиться указать маленькую матрицу типа $\binom{1\,1}{0\,1}$, которую можно ввести, например, так: `$1\,1\choose0\,1$`. Аналогично, маленькая матрица $\left(\begin{smallmatrix} a & b & c \\ l & m & n \end{smallmatrix}\right)$ может быть введена так:

```
$\bigl({a\atop l}{b\atop m}{c\atop n}\bigr)$
```

Макрокоманда `\matrix` не производит таких маленьких массивов.

9. Вертикальные пробелы. Если вы хотите привести в порядок необычную формулу, вы уже знаете, как раздвинуть элементы и подвинуть их ближе друг к другу, используя положительные и отрицательные тонкие пробелы. Но такие пробелы действуют только на горизонтальные размеры. А если вы хотите подвинуть что-нибудь выше или ниже? Это уже более сложная тема.

Приложение B предусматривает несколько макрокоманд, которые могут быть использованы, чтобы обмануть TEX и заставить его думать, что некоторые формулы больше или меньше, чем они есть в действительности. Такие трюки могут быть использованы, чтобы подвинуть какие-то части формул вверх, вниз, влево или вправо. Например, мы уже обсуждали `\mathstrut` в главе 16 и `\strut` в главе 17. Эти невидимые боксы указывают TEX'у ставить знаки квадратных корней и знаменатели цепных дробей не в такое положение, как обычно.

Если вы в какой-нибудь формуле говорите `\phantom{`⟨подформула⟩`}`, TEX сделает все пробелы так, как если бы вы просто сказали `{`⟨подформула⟩`}`, но сама подформула будет невидимой. Так, например, `\phantom{0}2` занимает в точности столько же места, сколько '`02`' в текущем стиле, но в действительности на странице появится только `2`. Если вы хотите оставить пустое место для нового символа, который имеет в точности такой же размер, как $\sum$, но по каким-то причинам вынуждены вставить этот символ вручную, то `\mathop{\phantom\sum}` оставит пустое место в точности нужной величины. (Здесь из-за `\mathop` фантом ведет себя так же, как `\sum`, т.е., как большой оператор.)

Еще более полезным, чем `\phantom`, является `\vphantom`, который создает такой невидимый бокс, высота и глубина которого равна высоте и глубине соответствующего `\phantom`, а ширина равна нулю. Таким образом, `\vphantom` создает вертикальную подпорку,которая может увеличить реальную высоту или глубину формулы. Начальный TEX определяет `\mathstrut`, как сокращение для '`\vphantom(`'. Существует также `\hphantom`, который имеет такую же ширину, как `\phantom`, но его высота и глубина равны нулю.

Начальный TEX также предусматривает `\smash{`⟨подформула⟩`}`, макрокоманду, которая дает тот же результат, что и `{`⟨подформула⟩`}`, но делает высоту и глубину равными нулю. Используя как `\smash`, так и `\vphantom`, можно напечатать любую подформулу и задать ей любые желаемые неотрицательные высоту и глубину. Например,

```
\mathop{\smash\limsup\vphantom\liminf}
```

дает большой оператор, который говорит $\limsup$, но его высота и глубина такие же, как у `\liminf` (т.е., глубина равна нулю).

▸ **Упражнение 18.26**
Если надо подчеркнуть некоторый текст, то можно использовать макрокоманду

```
\def\undertext#1{$\underline{\hbox{#1}}$}
```

Но это не всегда хорошо работает. Обсудите более хорошие альтернативы.

Чтобы регулировать вертикальное положение боксов в формулах, можно также использовать `\raise` и `\lower`. Например, в формуле

```
$2^{\raise1pt\hbox{$\scriptstyle n$}}$
```

верхний индекс n будет на 1pt выше, чем обычно (2^n вместо 2^n). Заметим, что в этом примере было необходимо сказать `\scriptstyle`, поскольку содержимое `\hbox` обычно бывает в текстовом стиле, даже когда этот h-бокс появляется в верхнем индексе, и поскольку `\raise` можно использовать только вместе с боксом. Этот метод позиционирования используется не слишком часто, но он иногда полезен, если макрокоманда `\root` не помещает аргумент на подходящее место. Например,

```
\root\raise⟨размер⟩\hbox{$\scriptscriptstyle⟨аргумент⟩$}\of...
```

будет поднимать аргумент на заданную величину.

Вместо изменения размеров подформул или использования `\raise`, можно управлять вертикальными пробелами, меняя параметры, которые TEX использует, когда преобразует математические списки в горизонтальные. Эти параметры описаны в приложении G. Но будьте внимательны, когда их меняете, поскольку эти изменения являются глобальными (т.е., не локальными в группах). Приведем пример того, как могло быть сделано такое изменение. Предположим, что вы разрабатываете формат для печати химических работ, и ожидаете, что будет много формул типа $\mathrm{Fe}_2^{+2}\mathrm{Cr}_2\mathrm{O}_4$. Вам может не понравиться тот факт, что нижний индекс в Fe_2^{+2} расположен ниже, чем нижний индекс в Cr_2, и не хочется вынуждать пользователя печатать монстров типа

```
$\rm Fe_2^{+2}Cr_2^{\vphantom{+2}}O_4^{\vphantom{+2}}$
```

только для того, чтобы получить формулу $\mathrm{Fe}_2^{+2}\mathrm{Cr}_2\mathrm{O}_4$ со всеми индексами на одном уровне. Тогда все, что нужно — это установить `\fontdimen16\tensy=2.7pt` и `\fontdimen17\tensy=2.7pt`, предполагая, что `\tensy` — это ваш основной символьный шрифт (`\textfont2`). Это опускает все нормальные нижние индексы в положение на 2.7 pt ниже базовой линии, которое достаточно для того, чтобы оставить место для возможного верхнего индекса, содержащего знак плюс. Аналогично можно регулировать положение верхнего индекса, изменяя `\fontdimen14\tensy`. Это параметры для положения осевой линии, положения числителя и знаменателя в обобщенной дроби, пробелов над и под пределами, толщины черты, принятой по умолчанию и т.д. Приложение G содержит точные детали.

10. Специальные возможности для математических хакеров. TEX имеет еще несколько примитивных операций для математической моды, которые еще не упоминались и которые могут оказаться полезными, когда разрабатываются специальные форматы.

Если перед спецификациями клея или керна находится `\nonscript`, TEX не будет использовать этот клей или керн в индексном или повторном индексном стилях. Так, например, команда `\nonscript\;` дает величину пробела, указанного как `(3)` в таблице пробелов для математики, приведенной в этой главе ранее.

Как только TEX прочитывает `$` и готовится читать математическую формулу, которая встречается в тексте, он сначала будет читать другой список элементов, предварительно определенный командой

`\everymath={`⟨список элементов⟩`}`

(Это аналог `\everypar`, описанного в главе 14.) Аналогично, чтобы предопределить список элементов, который TEX прочтет сразу после открывающих `$$`, т.е., перед чтением выделяемой формулы, можно сказать

`\everydisplay={`⟨список элементов⟩`}`

При помощи `\everymath` и `\everydisplay` можно установить специальные соглашения, которые применяются ко всем формулам.

11. Резюме. В этой главе мы обсудили больше различных видов формул, чем можно найти в любой книге по математике. Если вы до сих пор добросовестно выполняли упражнения, то можете смело встретиться лицом к лицу с любой формулой.

Приведем еще несколько упражнений, которые помогут вам повторить то, что вы выучили. Каждая из следующих "тестовых формул" иллюстрирует один или более принципов, которые уже обсуждались в этой главе. Автор признается, что он попытается поймать вас на некоторых из них. Однако, если вы попробуете выполнить каждое упражнение перед тем, как заглянуть в ответ, и будете внимательны к ловушкам, то обнаружите, что эти формулы помогли вам собрать и завершить ваши знания.

▸ **Упражнение 18.27**
Тест номер 1: Объясните, как напечатать фразу "$n^{\text{ый}}$ корень", где "$n^{\text{ый}}$" трактуется как математическая формула с верхним индексом в прямом шрифте.

▸ **Упражнение 18.28**
Тест номер 2: $\mathbf{S}^{-1}\mathbf{T}\mathbf{S} = \mathbf{dg}(\omega_1, \ldots, \omega_n) = \mathbf{\Lambda}$.

▸ **Упражнение 18.29**
Тест номер 3: $\Pr(\, m = n \mid m + n = 3\,)$.

▸ **Упражнение 18.30**
Тест номер 4: $\sin 18^\circ = \frac{1}{4}(\sqrt{5} - 1)$.

▸ **Упражнение 18.31**
Тест номер 5: $k = 1.38 \times 10^{-16}\,\mathrm{erg}/^\circ\mathrm{K}$.

▸ **Упражнение 18.32**
Тест номер 6: $\bar{\Phi} \subset NL_1^*/N = \bar{L}_1^* \subseteq \cdots \subseteq NL_n^*/N = \bar{L}_n^*$.

▸ **Упражнение 18.33**
Тест номер 7: $I(\lambda) = \iint_D g(x,y) e^{i\lambda h(x,y)}\, dx\, dy$.

▸ **Упражнение 18.34**
Тест номер 8: $\int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_n)\, dx_1 \ldots dx_n$.

▸ **Упражнение 18.35**
Тест номер 9: Вот выделенная формула

$$x_{2m} \equiv \begin{cases} Q(X_m^2 - P_2 W_m^2) - 2S^2 & (m \text{ нечетное}) \\ P_2^2(X_m^2 - P_2 W_m^2) - 2S^2 & (m \text{ четное}) \end{cases} \pmod N.$$

▸ **Упражнение 18.36**
Тест номер 10: И еще одна:

$$(1 + x_1 z + x_1^2 z^2 + \cdots) \ldots (1 + x_n z + x_n^2 z^2 + \cdots) = \frac{1}{(1 - x_1 z) \ldots (1 - x_n z)}.$$

▸ **Упражнение 18.37**
Тест номер 11: И еще одна:

$$\prod_{j \ge 0} \Bigl(\sum_{k \ge 0} a_{jk} z^k \Bigr) = \sum_{n \ge 0} z^n \Biggl(\sum_{\substack{k_0, k_1, \ldots \ge 0 \\ k_0 + k_1 + \cdots = n}} a_{0k_0} a_{1k_1} \ldots \Biggr).$$

▸ **Упражнение 18.38**
Тест номер 12: И такая:

$$\frac{(n_1 + n_2 + \cdots + n_m)!}{n_1!\, n_2! \ldots n_m!} = \binom{n_1 + n_2}{n_2} \binom{n_1 + n_2 + n_3}{n_3} \cdots \binom{n_1 + n_2 + \cdots + n_m}{n_m}.$$

▸ **Упражнение 18.39**
Тест номер 13: Еще одна выделенная формула:

$$\Pi_R \begin{bmatrix} a_1, a_2, \ldots, a_M \\ b_1, b_2, \ldots, b_N \end{bmatrix} = \prod_{n=0}^{R} \frac{(1 - q^{a_1 + n})(1 - q^{a_2 + n}) \ldots (1 - q^{a_M + n})}{(1 - q^{b_1 + n})(1 - q^{b_2 + n}) \ldots (1 - q^{b_N + n})}.$$

▸ **Упражнение 18.40**
Тест номер 14: И другая:

$$\sum_{p \text{ prime}} f(p) = \int_{t > 1} f(t)\, d\pi(t).$$

▸ **Упражнение 18.41**
Тест номер 15: Еще другая:

$$\underbrace{\{\overbrace{a, \ldots, a}^{k\ a\text{'s}}, \overbrace{b, \ldots, b}^{l\ b\text{'s}}\}}_{k+l \text{ elements}}.$$

▸ **Упражнение 18.42**
Тест номер 16: Поставьте `\smallskip` между рядами матриц в составной матрице

$$\begin{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} & \begin{pmatrix} e & f \\ g & h \end{pmatrix} \\ 0 & \begin{pmatrix} i & j \\ k & l \end{pmatrix} \end{pmatrix}.$$

▸ **Упражнение 18.43**
Тест номер 17: Сделайте здесь колонки прижатыми влево:

$$\det\begin{vmatrix} c_0 & c_1 & c_2 & \dots & c_n \\ c_1 & c_2 & c_3 & \dots & c_{n+1} \\ c_2 & c_3 & c_4 & \dots & c_{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & c_{n+2} & \dots & c_{2n} \end{vmatrix} > 0.$$

▸ **Упражнение 18.44**
Тест номер 18: Основная проблема здесь — это взять производную от $\sum$:

$$\sum_{x\in A}{}' f(x) \stackrel{\text{def}}{=} \sum_{\substack{x\in A\\ x\neq 0}} f(x).$$

▸ **Упражнение 18.45**
Тест номер 19: Может быть, теперь вы готовы к такой выделенной формуле:

$$2 \uparrow\uparrow k \stackrel{\text{def}}{=} \Big\}k \text{ раз}\Big\{ 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}.$$

▸ **Упражнение 18.46**
Тест номер 20: И наконец, когда вы разделались со всеми другими примерами, приведем окончательный тест. Объясните, как получить коммутативную диаграмму:

$$\begin{array}{ccccccccc} & & & & & & 0 & & \\ & & & & & & \downarrow & & \\ 0 & \longrightarrow & \mathcal{O}_C & \xrightarrow{\iota} & \mathcal{E} & \xrightarrow{\rho} & \mathcal{L} & \longrightarrow & 0 \\ & & \| & & \downarrow{\scriptstyle\phi} & & \downarrow{\scriptstyle\psi} & & \\ 0 & \longrightarrow & \mathcal{O}_C & \longrightarrow & \pi_*\mathcal{O}_D & \xrightarrow{\delta} & R^1 f_*\mathcal{O}_V(-D) & \longrightarrow & 0 \\ & & & & & & \downarrow{\scriptstyle\theta_i\otimes\gamma^{-1}} & & \\ & & & & & & R^1 f_*\big(\mathcal{O}_V(-iM)\big)\otimes\gamma^{-1} & & \\ & & & & & & \downarrow & & \\ & & & & & & 0 & & \end{array}$$

используя `\matrix`. (Многие элементы пустые.)

12. Маленький совет. Число различных типов записей огромно и все еще растет, поэтому в процессе ввода математических документов вы будете встречать все новые и новые тесты. Хорошая идея завести персональную тетрадку, в которую записывать все неочевидные формулы, с которыми вы

успешно справились, показывая как окончательный результат, так и то, что надо вводить, чтобы его получить. Поэтому через несколько месяцев, когда потребуется сделать что-нибудь аналогичное, вы можете использовать те решения, которые уже нашли.

Если вы математик, который печатает свои собственные статьи, вы теперь выучили, как получить самые сложные формулы и можете это делать, не обращаясь к посреднику, который может как-нибудь исказить их значение. Но пожалуйста, не слишком увлекайтесь своим новоиспеченным талантом. Тот факт, что вы при помощи TEX'а можете напечатать формулы, не обязательно означает, что вы нашли наилучшую запись для контакта с читателями своей работы. Некоторые варианты записи будут неудачными, даже хотя они прекрасно отформатированы.

Математики — как французы:
стоит им что-нибудь сказать, как они
сразу же переводят это на свой язык,
и смысл полностью меняется.

Mathematicians are like Frenchmen:
whenever you say something to them,
they translate it into their own language,
and at once it is something entirely different.

— GOETHE, *Maxims and Reflexions* (1829)

Лучшие обозначение — это отсутствие обозначений;
если можно обойтись без
сложного буквенного аппарата,
обходитесь без него.
Лучше всего при подготовке написанного
математического изображения представить,
что вы его рассказываете.
Представьте, что вы объясняете предмет
приятелю во время прогулки в лесу,
когда под рукой нет ни клочка
бумаги; прибегайте к символам
только при крайней
необходимости

The best notation is no notation;
whenever it is possible to avoid the use
of a complicated alphabetic apparatus,
avoid it.
A good attitude to the preparation
of written mathematical exposition
is to pretend that it is spoken.
Pretend that you are explaining
the subject to a friend
on a long walk in the woods,
with no paper available;
fall back on symbolism only
when it is really necessary.

— PAUL HALMOS, *How to Write Mathematics* (1970)

19

Выделенные уравнения

Сейчас вы уже знаете, как вводить математические формулы, чтобы TEX обращался с ними наиболее элегантно, и изучение математического набора почти завершено. Но осталась еще одна сторона дела, и эта глава служит его счастливому завершению. Мы обсудили, как обходиться с отдельными формулами, но часто выделяются целые букеты различных формул или различные куски громадной формулы. Проблема состоит в том, чтобы расположить их так, чтобы они правильно выстроились одна под другой. К счастью, большие формулы обычно распадаются на несколько простых элементов.

1. Однострочные выделенные формулы. Прежде чем погрузиться в общие вопросы расположения выделенных формул, давайте резюмируем то, что мы уже выяснили. Если вы вводите `$$`⟨формула⟩`$$`, TEX напечатает эту формулу в выделенном стиле, центрируя ее на строке. Мы также отметили в главе 18, что можно за один раз напечатать две короткие формулы, вводя `$$`⟨формула$_1$⟩`\qquad`⟨формула$_2$⟩`$$` — это превращает проблему двух формул в проблему одной формулы. Вы получаете две формулы, разделенные двумя квадратами пробельного материала и в целом центрированные на строке.

Выделенные уравнения часто включают обычный текст. Глава 18 объясняет, как получить прямой шрифт в формулах, не выходя из математической моды, но лучше получать текст в выделенных формулах, помещая его в `\hbox`. Тут вообще не требуется никакой математики. Для того, чтобы напечатать

Выделенный Текст

можно просто сказать `$$\hbox{Выделенный Текст}$$`. Но приведем более интересный пример:

$$X_n = X_k \qquad \text{тогда и только тогда, когда} \qquad Y_n = Y_k \quad \text{и} \quad Z_n = Z_k.$$

В этом случае формулы и текст скомбинированы так:

```
$$X_n=X_k \qquad\hbox{тогда и только тогда, когда}\qquad
    Y_n=Y_k \quad\hbox{и}\quad Z_n=Z_k.$$
```

Заметим, что “тогда и только тогда, когда” окружено `\qquad`, а “и” окружено `\quad`. Это помогает указать, что части Y и Z выделенной формулы связаны более тесно друг с другом, чем с частью X.

Рассмотрим теперь выделенные формулы

$$Y_n = X_n \bmod p \quad \text{и} \quad Z_n = X_n \bmod q \qquad \text{для всех } n \ge 0.$$

Можете догадаться, как их ввести? Одним из решений является

```
$$Y_n=X_n\bmod p \quad\hbox{и}\quad Z_n=X_n\bmod q
    \qquad\hbox{для всех }n\ge0.$$
```

Заметим, что после “`всех`” в h-боксе оставлен пробел, поскольку пробелы, когда они оказываются на территории формул, исчезают. Но существует

более простой и более логичный способ действий, если вы освоились с концепцией мод TEX'а. Можно ввести

```
... \qquad\hbox{для всех $n\ge0$.}$$
```

Поразительно — математическая мода внутри горизонтальной моды внутри выделенной математической моды! Но зато в этом случае рукопись наглядно отражает то, что вы пытаетесь сделать, в то время как предыдущее решение (с пробелом после "`всех`") выглядит несколько натянуто.

▸ **Упражнение 19.1**
Напечатайте следующие четыре выделенные формулы (по одной):

$$\sum_{n=0}^{\infty} a_n z^n \qquad \text{сходится, если} \qquad |z| < \Bigl(\limsup_{n\to\infty} \sqrt[n]{|a_n|}\Bigr)^{-1}.$$

$$\frac{f(x+\Delta x)-f(x)}{\Delta x} \to f'(x) \qquad \text{при } \Delta x \to 0.$$

$$\|u_i\| = 1, \qquad u_i \cdot u_j = 0 \quad \text{если } i \ne j.$$

$$\textit{Конфлуентный образ} \left\{\begin{array}{l}\textit{дуги}\\ \textit{окружности}\\ \textit{сектора}\end{array}\right\} \textit{ есть } \left\{\begin{array}{l}\textit{дуги}\\ \textit{дуга или окружность}\\ \textit{сектор или дуга}\end{array}\right\}.$$

▸ **Упражнение 19.2**
Иногда формула в выделенном стиле оказывается слишком крупной, например, формула

$$y = \frac{1}{2}x$$

или что-нибудь столь же простое. Однажды B. L. User попытался исправить это, напечатав эту формулу как `$$y={\scriptstyle1\over\scriptstyle2}x$$`, но полученная формула

$$y = {}^{1}\!/_{2}\,x$$

оказалась не совсем такой, какую он имел ввиду. Как правильно получить просто $y = \frac{1}{2}x$, если вы не хотите, чтобы выделенные дроби получились большими?

▸ **Упражнение 19.3**
Какое различие, если оно есть, между результатами `$$`⟨формула⟩`$$` и `$$\hbox{$`⟨формула⟩`$}$$`?

▸ **Упражнение 19.4**
Вы могли заметить, что большинство выделенных формул в этом руководстве не центрированы. По дизайну выделенный материал выровнен слева по отступу абзаца, поскольку это необычная книга. Объясните, как бы вы могли напечатать формулу

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots = \ln 2$$

которая сдвинута от центра таким же образом.

Если у вас уже был опыт набора математических текстов, вы, вероятно, задаете вопрос: "А что делать с номерами уравнений? Где эта книга собирается рассказать о них?" Вы правы, теперь самое время обсудить эти незаметные маленькие метки, которые появляются сбоку от выделенных формул. Если вы введете

```
$$⟨формула⟩\eqno⟨формула⟩$$
```

TEX выделит первую формулу, а около правого поля поставит номер уравнения (вторую формулу). Например,

```
$$x^2-y^2 = (x+y)(x-y).\eqno(15)$$
```

дает:

$$x^2 - y^2 = (x + y)(x - y). \tag{15}$$

Также можно получить номер уравнения у левого поля, используя `\leqno`. Например,

```
$$x^2-y^2 = (x+y)(x-y).\leqno(16)$$
```

дает:

(16) $$x^2 - y^2 = (x + y)(x - y).$$

Заметим, что номер уравнения всегда задается вторым, даже когда ему следует появиться слева. Все, начиная от команд `\eqno` или `\leqno` и до `$$`, которым оканчивается выделение, представляет собой номер уравнения. Таким образом, нельзя в одном и том же выделении иметь два номера уравнения, но, как мы увидим позже, есть способ обойти это ограничение.

В настоящее время все больше и больше используются номера уравнений, расположенные справа, потому что выделенные формулы часто располагаются в конце предложения, а правостороннее соглашение отделяет номер от текста предложения. Более того, часто появляется возможность сэкономить место, если выделенное уравнение следует за короткой текстовой строкой, поскольку тогда нужно меньше места над выделенной формулой. Такая экономия невозможна с `\leqno`, поскольку в этом случае нет места для частичного перекрывания. Например, в наших иллюстрациях `\eqno` и `\leqno` над уравнением (15) меньше места, чем над уравнением (16), хотя в остальных отношениях формулы и текст абсолютно идентичны.

Если вы внимательно посмотрите выше на (15) и (16), то сможете увидеть, что выделенные формулы центрированы без учета номера уравнения. Но когда формула велика, TEX делает так, что она не мешает своему номеру. Номер уравнения может быть даже помещен на отдельной строке.

▸ **Упражнение 19.5**
Как можно получить следующую выделенную формулу?

$$\prod_{k\ge0}\frac{1}{(1-q^kz)} = \sum_{n\ge0} z^n \Big/ \prod_{1\le k\le n}(1-q^k). \tag{16′}$$

▸ **Упражнение 19.6**
Номера уравнений — это математические формулы, набранные в текстовом стиле. Итак, как вы можете получить такой номер, как (3–1) (с дефисом)?

▸ **Упражнение 19.7**
B. L. User попытался напечатать `\eqno(*)` и `\eqno(**)` и с удовольствием обнаружил, что это дало номера уравнений (∗) и (∗∗). [Он немного волновался, что вместо этого могло получится (*) и (**).] Но затем несколькими месяцами позже он попробовал `\eqno(***)` и получил сюрприз. Какой?

В этом руководстве обязательно должно быть приведено точное описание того, как TEX выделяет формулы, т.е., как он центрирует их, как размещает номера уравнений, как вставляет дополнительные пробелы над или под формулой и т.д. Теперь настало время определить эти правила. Они довольно-таки сложны, поскольку взаимодействуют с элементами типа `\parshape` и включают в себя несколько параметров, которые еще не обсуждались. Цель этих правил — точно объяснить, какие виды боксов, клея и штрафов помещаются в текущий вертикальный список, когда встречается выделенная формула.

Если выделенная формула встречается после, скажем, четырех строк абзаца, то, когда начинается выделение, внутренний регистр TEX'а `\prevgraf` будет равен 4. Три строки добавит выделение, так что, когда абзац после конца выделения возобновится, `\prevgraf` станет равным 7 (если только вы за это время не изменили `\prevgraf`). TEX присваивает специальные значения трем параметрам типа ⟨размер⟩ сразу после того, как прочитаны открывающие `$$`: `\displaywidth` и `\displayindent` устанавливаются равными ширине строки z и величине сдвига s для строки номер `\prevgraf` + 2, на основании текущей формы абзаца и подвешенного отступа. (Обычно `\displaywidth` равно `\hsize`, а `\displayindent` равно нулю, но форма абзаца может изменяться, как описано в главе 14.) Более того, `\predisplaysize` устанавливается равным действительной ширине p строки, предшествующей выделению; если нет предыдущей строки (например, если перед `$$` следует `\noindent` или закрывающие `$$` другого выделения), p устанавливается равным −16383.99999 pt (т.е., наименьшему разрешенному размеру −`\maxdimen`). В противном случае TEX смотрит внутрь h-бокса, который сформирован предыдущей строкой и устанавливает p в положение правого края самого правого бокса внутри этого h-бокса, плюс отступ, на который этот внешний h-бокс сдвинут вправо, плюс 2 em текущего шрифта. Однако, если это значение p зависит от того, что клей в этом h-боксе растягивался или сжимался, например, если клей `\parfillskip` конечен, так что материал, предшествующий ему, не установлен в свою натуральную ширину, то p устанавливается равным `\maxdimen`. (Это не часто случается, но делает TEX машинно-независимым, поскольку p никогда не зависит от величин, которые могут быть по-разному округлены на различных компьютерах.) Заметим, что на `\displaywidth` и `\displayindent` не влияют `\leftskip` и `\rightskip`, но влияет `\predisplaysize`. Значения `\displaywidth`, `\displayindent` и `\predisplaysize` будут использованы TEX'ом после того, как он прочтет выделенную формулу, как это объясняется ниже. Ваша программа может проверить и/или изменить их, если вы хотите печатать что-нибудь другим способом.

После того, как прочитана выделенная формула, TEX преобразует ее из математического списка в горизонтальный список h в выделенном стиле,

как объяснено в приложении G. Номер уравнения, если он присутствует, переводится в текстовый стиль и помещается в h-бокс a со своей натуральной шириной. Теперь начинается тщательная обработка. Пусть z, s и p — текущие значения `\displaywidth`, `\displayindent` и `\predisplaysize`. Пусть q и e равны нулю, если нет номера уравнения, а если есть номер, пусть e — ширина номера уравнения, и пусть q равна e плюс один квадрат в символьном шрифте (т.е., в `\textfont2`). Пусть w_0 — натуральная ширина выделенной формулы h. Если $w_0 + q \le z$, то список h упаковывается в h-бокс b, имеющий натуральную ширину w_0. Но если $w_0 + q > z$ (т.е., если выделенная формула слишком широка, чтобы поместиться в свою натуральную ширину), TEX выполняет следующую "сжимающую программу": если $e \neq 0$ и если в выделенной формуле h достаточно сжимаемости, чтобы сократить ее ширину до $z - q$, список h упаковывается в h-бокс b шириной $z - q$. В противном случае e устанавливается равным нулю, и список h упаковывается в (возможно, переполненный) h-бокс b шириной $\min(w_0, z)$.


(Продолжение.) Теперь TEX пытается центрировать формулу без номера уравнения. Но если такое центрирование выполнено слишком близко к этому номеру (где "слишком близко" означает, что расстояние между ними меньше, чем ширина e), уравнение либо центрируется на оставшемся месте, либо помещается насколько возможно далеко от своего номера. Последняя альтернатива выбирается только тогда, когда первым элементом списка h является клей, поскольку TEX предполагает, что такой клей был там помещен для точного управления пробелами. Но давайте сформулируем правила более формально. Пусть w — ширина бокса b. TEX вычисляет смещение d, чтобы использовать его позже, когда он размещает бокс b, сначала установив $d = \frac{1}{2}(z - w)$. Если $e > 0$ и если $d < 2e$, то d переопределяется в $\frac{1}{2}(z - w - e)$ или в ноль, где ноль выбирается, если h начинается с клея.


(Продолжение.) Теперь TEX готов размещать элементы в вертикальном списке сразу после материала, который был получен для предыдущей части абзаца. Сначала следует элемент штрафа, стоимость которого равна целому параметру, называемому `\predisplaypenalty`. Затем следует клей. Если $d + s \le p$ или если есть левый номер уравнения (`\leqno`), TEX устанавливает g_a и g_b как элементы клея, заданные, соответственно, параметрами `\abovedisplayskip` и `\belowdisplayskip`; в противном случае g_a и g_b становятся элементами клея, соответствующими `\abovedisplayshortskip` и `\belowdisplayshortskip`. [Пояснение: если предвыделенный размер достаточно короткий, чтобы не перекрывать выделенную формулу, т. е., последняя строчка не наползает на формулу, то клей выше и ниже выделения будет "короче" по сравнению с клеем, который используется, когда имеет место перекрывание.] Если $e = 0$ и если есть `\leqno`, то номер уравнения добавляется как h-бокс, который сдвинут вправо на s и которому, как обычно, предшествует междустрочный клей. Кроме того, задается бесконечный штраф, чтобы предотвратить разбиение страницы между этим номером и выделенной формулой. В противном случае в вертикальный список помещается элемент клея g_a.


(Продолжение.) Теперь идет само выделенное уравнение. Если $e \neq 0$, бокс номера уравнения a комбинируется с боксом формулы b следующим образом. Пусть k — керн шириной $z - w - e - d$. В случае `\eqno` бокс b замещается

h-боксом, содержащим (b, k, a), а в случае `\leqno` бокс b замещается h-боксом, содержащим (a, k, b) и d устанавливается равным нулю. Во всех случаях бокс b затем присоединяется к вертикальному списку со сдвигом вправо на $s + d$.


(Продолжение.) Конечная задача — это присоединить клей или номер уравнения, который следует за выделением. Если есть `\eqno` и если $e = 0$, то в вертикальный список помещается бесконечный штраф, за которым следует бокс номера уравнения a, сдвинутый вправо на $s + z$ минус его ширина, а за ним штраф, стоимость которого равна значению `\postdisplaypenalty`. В противном случае штраф для `\postdisplaypenalty` присоединяется первым, а за ним, как указывалось выше, клей для g_b. TEX теперь к `\prevgraf` добавляет 3 и возвращается в горизонтальную моду, готовый продолжить абзац.


Эти правила имеют такое следствие: можно заставить номер уравнения появиться на отдельной строке, сделав его ширину нулевой, т.е., сказав либо `\eqno\llap{$`⟨формула⟩`$}`, либо `\leqno\rlap{$`⟨формула⟩`$}`. Это делает $e = 0$, а условие $e = 0$ управляет логикой TEX'а при позиционировании, как это объяснялось в только что приведенных правилах.


Начальный TEX устанавливает `\predisplaypenalty=10000`, потому что наборщики с тонким вкусом традиционно избегают выделенных формул вверху страницы. Можно изменить `\predisplaypenalty` и `\postdisplaypenalty`, если вы хотите поощрить или избежать разбиения страницы непосредственно перед или после выделения. Например, `$$\postdisplaypenalty=-10000`⟨формула⟩`$$` приведет к разбиению страницы, поставив формулу на верхнюю строку. Лучше разбить страницу таким способом, чем сказать `\eject` сразу после `$$...$$`. Такой сброс (который следует за клеем `\belowdisplayskip` под формулой) заставляет страницу быть короткой, поскольку оставляет нежелательный клей внизу.


▸ **Упражнение 19.8**
Прочитайте внимательно правила и определите окончательное расположение $x = y$ в формуле

```
$$\quad x=y \hskip10000pt minus 1fil \eqno(5)$$
```

предполагая, что там нет подвешенного отступа. Также рассмотрите `\leqno` вместо `\eqno`.


TEX также позволяет "выровненные выделения", которые не обрабатываются в математической моде, поскольку не содержат формул на внешнем уровне. Выровненное выделение создается командами следующего общего вида:

`$$`⟨назначения⟩`\halign{`⟨выравнивание⟩`}`⟨назначения⟩`$$`,

где ⟨назначения⟩ — это необязательные элементы типа изменения параметров, которые не производят каких-либо математических списков. В таких выделениях команда `\halign` работает в точности так же, как если бы она появилась в вертикальной моде и, как обычно, создает вертикальный список v, за тем исключением, что каждый ряд выравнивания будет сдвинут вправо при помощи `\displayindent`. После выравнивания и после того, как будут выполнены закрывающие назначения, TEX помещает в основной вертикальный список элемент `\predisplaypenalty` и некоторый клей `\abovedisplayskip`, за ним v, далее элемент `\postdisplaypenalty` и

клей `\belowdisplayskip`. Таким образом, выровненные выделения, по существу, похожи на обычные выравнивания, за исключением того, что они могут прерывать абзац. Более того, они окаймлены клеем и штрафами также, как другие выделения. Параметры `\displaywidth` и `\predisplaysize` не влияют на результат, хотя их можно использовать в `\halign`. Таким образом, целое выровненное выделение состоит только из трех строк, поскольку связано с `\prevgraf`.

2. Многострочные формулы. О'кей, очень приятно использовать выделенные формулы. Но когда вам придется часто набирать рукописи, вы будете сталкиваться с такими выражениями, которые не могут быть напечатаны как простая однострочная формула с или без номера уравнения. Начальный TEX предусматривает специальные команды, которые охватывают большинство оставшихся случаев.

Многострочные формулы обычно состоят из нескольких уравнений, которые должны быть выстроены по их знакам =, как в

$$X_1 + \cdots + X_p = m,$$
$$Y_1 + \cdots + Y_q = n.$$

Для этого рекомендуется следующая процедура: используйте `\eqalign`, работающую со специальными маркерами `&`, и `\cr`, с которыми мы уже сталкивались в связи с `\cases` и `\matrix` в главе 18. Покажем, как вводить этот частный случай:

```
$$\eqalign{X_1+\cdots+X_p&=m,\cr
           Y_1+\cdots+Y_q&=n.\cr}$$
```

В команде `\eqalign` может быть любое количество уравнений. Общий ее вид такой:

```
\eqalign{⟨левая часть1⟩&⟨правая часть1⟩\cr
         ⟨левая часть2⟩&⟨правая часть2⟩\cr
              ⋮
         ⟨левая частьn⟩&⟨правая частьn⟩\cr}
```

где каждая ⟨правая часть⟩ начинается с символа, по которому должно произойти выравнивание. Например, часто правая часть начинается со знака =. Уравнения будут напечатаны в выделенном стиле.

▸ **Упражнение 19.9**

На практике левые части выровненных формул часто бывают пустыми, а выравнивание производится как относительно символа =, так и относительно других символов. Например, типично следующее выделение. Попробуйте, если можете, угадать, как автор ввел его:

$$\begin{aligned} T(n) \le T(2^{\lceil \lg n \rceil}) &\le c(3^{\lceil \lg n \rceil} - 2^{\lceil \lg n \rceil}) \\ &< 3c \cdot 3^{\lg n} \\ &= 3c\, n^{\lg 3}. \end{aligned}$$

Результатом `\eqalign` является вертикально центрированный бокс. Это дает возможность легко получать такие формулы:

$$\left\{\eqalign{\alpha&=f(z)\cr \beta&=f(z^2)\cr \gamma&=f(z^3)\cr}\right\}\qquad\left\{\eqalign{x&=\alpha^2-\beta\cr y&=2\gamma\cr}\right\}.$$

Вы просто используете `\eqalign` дважды в одной и той же строке:

```
$$\left\{
\eqalign{\alpha&=f(z)\cr \beta&=f(z^2)\cr \gamma&=f(z^3)\cr}
\right\}\qquad\left\{
\eqalign{x&=\alpha^2-\beta\cr y&=2\gamma\cr}\right\}.$$
```

▸ **Упражнение 19.10**
Постарайтесь справиться с нумерованным двустрочным выделением:

$$\eqalign{P(x)&=a_0+a_1x+a_2x^2+\cdots+a_nx^n,\cr P(-x)&=a_0-a_1x+a_2x^2-\cdots+(-1)^na_nx^n.\cr}\tag{30}$$

[*Намек:* используйте тот факт, что `\eqalign` производит вертикально центрированный бокс; предполагается, что номер уравнения (30) должен появиться посередине между двумя строками.]

▸ **Упражнение 19.11**
Что случится, если вы в одном из уравнений в `\eqalign` забыли `&`?

Многострочные формулы иногда монтируются друг с другом необычным способом и оказывается, что время от времени вам хочется отодвинуть некоторые строки подальше от других или пододвинуть их ближе. Если после любого `\cr` вы введете `\noalign{\vskip`⟨клей⟩`}`, TEX после этой строки вставит заданную величину дополнительного клея. Например,

```
\noalign{\vskip3pt}
```

вставит между строками дополнительный пробел в 3 pt. Таким же способом можно изменить величину пробела перед первой строкой.

Со следующим уровнем сложности вы сталкиваетесь, когда у вас несколько выровненных уравнений и несколько номеров уравнений. Или, возможно, некоторые строки нумерованы, а другие — нет:

$$\eqalignno{(x+y)(x-y)&=x^2-xy+yx-y^2\cr &=x^2-y^2;&(4)\cr (x+y)^2&=x^2+2xy+y^2.&(5)\cr}$$

Для такой ситуации начальный TEX имеет команду `\eqalignno`; она используется также, как `\eqalign`, но на каждой строке, где нужен номер урав-

нения, добавляется `&`⟨номер уравнения⟩ непосредственно перед `\cr`. Приведенный выше пример был получен так:

```
$$\eqalignno{(x+y)(x-y)&=x^2-xy+yx-y^2\cr
    &=x^2-y^2;&(4)\cr
  (x+y)^2&=x^2+2xy+y^2.&(5)\cr}$$
```

Заметим, что второй `&` опускается, если нет номера уравнения.

Имеется также команда `\leqalignno`, которая помещает эти номера уравнения слева. В этом случае лучше передвинуть (4) в начало первого из уравнений:

$$(x+y)(x-y) = x^2 - xy + yx - y^2 \tag{4}$$
$$= x^2 - y^2;$$
$$(x+y)^2 = x^2 + 2xy + y^2. \tag{5}$$

Хотя номера уравнений появляются слева, вводить их надо справа, точно также, как вы это делали с `\leqno`. Другими словами, чтобы получить предыдущее выделение, вы должны ввести:

```
$$\leqalignno{(x+y)(x-y)&...&(4)\cr...}$$
```

Предостережение. Как `\eqalignno`, так и `\leqalignno` центрируют группу уравнений, не учитывая ширину их номеров. Если уравнения или их номера получаются слишком широкими, они могут перекрываться, тем не менее сообщение об ошибке не выдается.

▸ **Упражнение 19.12**

Напечатайте следующие выделенные формулы:

$$\gcd(u,v) = \gcd(v,u); \tag{9}$$
$$\gcd(u,v) = \gcd(-u,v). \tag{10}$$

▸ **Упражнение 19.13**

А вот вам еще для тренировки:

$$\begin{aligned}\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy\\ &= \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r\,dr\,d\theta\\ &= \int_0^{2\pi}\left(-\frac{e^{-r^2}}{2}\Bigg|_{r=0}^{r=\infty}\right)d\theta\\ &= \pi. \end{aligned} \tag{11}$$

Хотя результаты деятельности команд `\eqalign` и `\eqalignno` выглядят почти одинаково, между ними существует фундаментальное различие: `\eqalign` создает простой вертикальный центрированный бокс, который не шире, чем надо, а `\eqalignno` образует набор строк, которые имеют полную ширину

выделения (достигающую любыми способами обоих полей). Так, например, в выделении вы можете использовать `\eqalign` несколько раз, но `\eqalignno` может появиться только единожды. Если вы попробуете использовать `\eqno` вместе с `\eqalign`, то получите приличный результат, но если вы попытаетесь использовать `\eqno` вместе с `\eqalignno`, то получите сообщение о какой-нибудь фатальной ошибке (ошибках).

Приложение B раскрывает, почему `\eqalign` и `\eqalignno` ведут себя поразному: `\eqalign` — это сокращение для `\vcenter{\halign{...}}`, в то время как `\eqalignno` — это сокращение для `\halign to\displaywidth{...}`; таким образом макрокоманда `\eqalignno` генерирует "выровненное выделение".

Это различие между `\eqalign` и `\eqalignno` имеет два интересных следствия. (1) Нельзя разорвать страницу на `\eqalign`, а на `\eqalignno` разрыв может быть сделан. Действительно вы можете *принудить* к разрыву страницы после конкретной строки, если вставите на строке `\noalign{\break}` после `\cr`, и предотвратите такой *разрыв*, если вставите `\noalign{\nobreak}`. Можно запретить *все* разрывы в `\eqalignno`, если заключить всю формулу в `\vbox`:

```
$$\vbox{\eqalignno{...}}$$
```

(2) Можно также вставить между двумя уравнениями строку текста, не потеряв выравнивание. Например, рассмотрим две выделенные формулы:

$$x = y + z$$

и

$$x^2 = y^2 + z^2.$$

Они были получены как одно выделение таким способом:

```
$$\eqalignno{x&=y+z\cr
  \noalign{\hbox{и}}
  x^2&=y^2+z^2.\cr}$$
```

Поэтому то, что знаки = в этих формулах оказались один под другим, не является счастливым совпадением. Иногда может потребоваться отрегулировать пробелы над или под такой строкой вставленного текста, вставляя один или два `\vskip` внутри `\noalign{...}`. Заодно этот пример показывает, что можно использовать `\eqalignno`, не задавая номеров уравнений.

▸ **Упражнение 19.14**
Что будет, если в последнем примере `\eqalign` заменить на `\eqalignno`?

▸ **Упражнение 19.15**
Наш друг Ben User снова попал в беду, когда попытался сдвинуть номер уравнения выше, чем обычно, написав так:

```
$$\eqalignno{...&\raise6pt\hbox{(5)}\cr}$$
```

Какую оплошность он совершил, и что можно было сделать вместо этого?

Для других типов выделения начальный TEX предусматривает команду `\displaylines`, которая позволяет выделить любое количество формул любым способом, который вам нравится, без какого-либо выравнивания. Общий вид команды такой:

$$\texttt{\$\$\textbackslash displaylines\{}\langle\text{выделяемая формула}_1\rangle\texttt{\textbackslash cr}$$
$$\langle\text{выделяемая формула}_2\rangle\texttt{\textbackslash cr}$$
$$\vdots$$
$$\langle\text{выделяемая формула}_n\rangle\texttt{\textbackslash cr\}\$\$}$$

Каждая формула будет центрирована, поскольку `\displaylines` ставит `\hfil` слева и справа от каждой строки. Можно подавить это центрирование, чтобы прижать формулу влево или вправо, вставив `\hfill`, которое имеет преимущество над `\hfil`.

▸ **Упражнение 19.16**
Воспользуйтесь `\displaylines`, чтобы ввести три выделенные формулы:

$$x \equiv x; \tag{1}$$
$$\text{если}\quad x \equiv y \quad \text{тогда}\quad y \equiv x; \tag{2}$$
$$\text{если}\quad x \equiv y \quad \text{и}\quad y \equiv z \quad \text{тогда}\quad x \equiv z. \tag{3}$$

Если вы внимательно посмотрите на многострочные выделения, приведенные в этой главе, то увидите, что базовые линии в них дальше друг от друга, чем в обычном тексте. Редакторы обычно делают это для того, чтобы сделать формулы более удобными для чтения. В соответствии с этой традицией `\eqalign` и ее родственники автоматически увеличивают `\baselineskip`. Если вы создаете многострочное выделение примитивной командой TEX'а `\halign` вместо того, чтобы употребить одну из макрокоманд начального TEX'а, вам тоже может понравиться такое регулирование базовых линий, и это легко сделать, сказав `$$\openup1\jot \halign{...}$$`. Макрокоманда `\openup` увеличивает `\lineskip` и `\lineskiplimit`, а также `\baselineskip`. Если вы говорите '`\openup2\jot`', строки раздвигаются на 2 дополнительные единицы. Начальный TEX использует единицы, равные 3 pt. Поскольку `$$...$$` действует как группа, действие `\openup` исчезает, когда оканчивается выделение. За `\openup` может следовать любой ⟨размер⟩, но принято выражать величину символически в терминах `\jot`, вместо того, чтобы использовать абсолютные единицы; тогда ваша рукопись может быть использована с множеством различных форматов.

Описания макрокоманд начального TEX'а `\displaylines`, `\eqalignno` и `\leqalignno` начинаются с `\openup1\jot`. Если не нужна такая раздвижка, можно удалить ее, сказав, например, `$$\openup-1\jot \eqalignno{...}$$`, поскольку `\openup` обладает эффектом накапливания.

Предположим, что вы решили сделать самодельное выделение, имеющее общий вид `$$\openup1\jot \halign{...}$$`, и для удобства давайте предположим, что действуют обычные соглашения начального TEX'а, так что `\jot=3pt` и `\baselineskip=12pt`. Тогда макрокоманда `\openup` заменяет расстояние между базовыми линиями на 15 pt. Отсюда следует, что базовая линия текстовой строки, которая предшествует выделению, будет выше самой верхней базовой линии

выделения на 15 pt плюс `\abovedisplayskip`. Но когда абзац возобновляется, следующая базовая линия будет ниже последней базовой линии выделения только на 12 pt плюс `\belowdisplayskip`, потому что параметру `\baselineskip` будет возвращено его нормальное значение. Для того, чтобы компенсировать это различие, макрокоманды `\eqalignno` и `\displaylines` перед первой строкой указывают `\noalign{\vskip`$-d$`}`, где d — чистая величина раздвижки.

3. Длинные формулы. Наше обсуждение математических текстов почти завершено. Осталось разобраться только с одной проблемой. Что делать, если формула настолько длинна, что не помещается в одну строку?

Например, предположим, что вам встретилось уравнение

$$\sigma(2^{34}-1,2^{35},1) = -3+(2^{34}-1)/2^{35}+2^{35}/(2^{34}-1)+7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).$$

Вам надо его как-то разделить. TEX постарался улучшить это уравнение, стиснув все вместе, сжимая пробелы около знаков + и − до нуля, но строка все еще оказывается переполненной.

Давайте попробуем разорвать это уравнение перед “+7”. Один из способов — это ввести

```
$$\eqalign{\sigma(2^{34}-1,2^{35},1)
&=-3+(2^{34}-1)/2^{35}+2^{35}\!/(2^{34}-1)\cr
&\qquad+7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\cr}$$
```

что дает

$$\begin{aligned}\sigma(2^{34}-1,2^{35},1) &= -3+(2^{34}-1)/2^{35}+2^{35}/(2^{34}-1)\\ &\qquad +7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\end{aligned}$$

Идея состоит в том, чтобы обрабатывать длинную однострочную формулу, используя `\qquad` на второй строке, так что вторая часть формулы появляется правее знака = на первой строке.

▸ **Упражнение 19.17**

Объясните, как справиться со следующей выделенной формулой:

$$\begin{aligned}x_nu_1+\cdots+x_{n+t-1}u_t &= x_nu_1+(ax_n+c)u_2+\cdots\\ &\qquad +\bigl(a^{t-1}x_n+c(a^{t-2}+\cdots+1)\bigr)u_t\\ &= (u_1+au_2+\cdots+a^{t-1}u_t)x_n+h(u_1,\ldots,u_t).\end{aligned}\tag{47}$$

Это настоящее искусство — решить, как разбить длинные выделенные формулы на несколько строк. TEX никогда не пытается разбивать их, поскольку нет адекватного набора правил. Автор математической рукописи обычно сам лучше может судить, как это сделать, поскольку место разбиения зависит от тонких факторов математического описания. Например, часто желательно выделить некоторые симметрии или другие структуры, которые лежат в основе формулы, а такие вещи требуют глубокого понимания того, что описывается в формуле.

Несмотря на это, есть возможность сформулировать несколько правил о том, как обращаться с длинными формулами в выделениях, поскольку существуют некоторые принципы, которым стараются следовать опытные математические наборщики.

a) Хотя формулы внутри абзаца всегда разрываются *после* бинарных операций и отношений, выделенные формулы всегда разрываются *перед* бинарными операциями и отношениями. Так, мы не должны оканчивать первую строку нашего $\sigma(\ldots)$-примера на "`(2^{34}-1)+`". Мы закончили ее на "`(2^{34}-1)`", и начали вторую строку с "`+`".

b) Когда уравнение разрывается перед бинарной операцией, вторая строка должна начинаться как минимум на два квадрата правее того места, где на первой строке начинается самая внутренняя подформула, содержащая этот знак бинарной операции. Например, если вы желаете разорвать

```
$$\sum_{0<k<n}\left(⟨формула1⟩+⟨формула2⟩\right)$$
```

на знаке плюс между ⟨формула$_1$⟩ и ⟨формула$_2$⟩, то почти обязательно, чтобы знак плюс на второй строке появился несколько правее большой левой скобки, которая соответствует "`\left(`".

В только что рассмотренном примере нужно специально позаботиться о разрыве формулы на две строки, так как ограничители `\left` и `\right` не могут использоваться изолировано: нельзя иметь на одной строке формулы только `\left`, а на второй строке — только `\right`. Более того, нужно, чтобы оба ограничителя были одинакового размера, несмотря на то, что они встречаются на разных строках. Обычно лучше выбрать размер ограничителя самому. Например, можно ввести

```
$$\eqalign{\sum_{0<k<n}\biggl(&⟨формула1⟩\cr
                &\qquad{}+⟨формула2⟩\biggr)\cr}$$
```

если ограничители `\bigg` являются самыми подходящими. Заметим, что в этом примере у знаков = не встречаются маркеры `&`, они отмечают только точку выравнивания.

Есть и другой способ разрывать длинные формулы, иногда называемый *двустрочной* формой. Идея состоит в том, чтобы поместить первую часть формулы почти вплотную слева, а вторую часть — почти вплотную справа, где "почти вплотную" означает: "отступив один квадрат". Так, двустрочная форма длинного уравнения $\sigma(\ldots)$, рассмотренного ранее, будет такова:

$$\sigma(2^{34}-1,2^{35},1) = -3+(2^{34}-1)/2^{35}+2^{35}/(2^{34}-1) \\ \qquad +7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).$$

Такой двустрочный эффект легко получить при помощи `\displaylines`:

```
$$\displaylines{\quad\sigma(2^{34}-1,2^{35},1)
  =-3+(2^{34}-1)/2^{35}+2^{35}\!/(2^{34}-1)\hfill\cr
\hfill{}+7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\quad\cr}$$
```

Здесь на второй строке напечатан дополнительный `{}`, так что TEX будет знать, что `+` — это бинарная операция. Двустрочная форма особенно рекомендуется для

уравнений, которые имеют длинную левую часть. В этом случае разрыв обычно оказывается прямо перед знаком =.

▸ **Упражнение 19.18**
Введите следующую выделенную формулу:

$$\sum_{1\le j\le n} \frac{1}{(x_j-x_1)\dots(x_j-x_{j-1})(x-x_j)(x_j-x_{j+1})\dots(x_j-x_n)} = \frac{1}{(x-x_1)\dots(x-x_n)}. \quad (27)$$

▸ **Упражнение 19.19**
Если необходимо ввести такую гигантскую дробь, как

$$\frac{q^{\frac12 n(n+1)}(ea;q^2)_\infty(eq/a;q^2)_\infty(caq/e;q^2)_\infty(cq^2/ae;q^2)_\infty}{(e;q)_\infty(cq/e;q)_\infty}$$

в одну узкую колонку, вам придется разорвать числитель и применить

$$\frac{q^{\frac12 n(n+1)}(ea;q^2)_\infty(eq/a;q^2)_\infty (caq/e;q^2)_\infty(cq^2/ae;q^2)_\infty}{(e;q)_\infty(cq/e;q)_\infty}$$

Как бы вы могли описать последнюю дробь на TEX'e?

Когда формула оказывается слишком длинной, и должна быть поделена на строки подходящей длины, (мы, конечно, говорим о выделенных формулах), ее, если можно, надо разбивать на концах естественных "фраз"; если, например, формула многоскобочная, ее следует разбивать на конце одного из главных выражений в скобках, а не на внутреннем символе. Эта естественная фразировка (как в музыке или в речи) служит взаимопониманию между автором и читателем и не должна передоверяться наборщику. Автор, когда ему приходится писать длинные формулы, должен указывать точки, удобные для разбиения, если оно окажется необходимым.

When a formula is too long for the page-width and has to be broken into successive lines (and we are now, of course, speaking of displayed formulae), it should be broken, if possible, at the end of a natural 'phrase'; if, for example, it is a much-bracketed formula, it should be broken at the end of one of the major brackets and not at an inner symbol. This natural phrasing (as in music or speech) makes for intelligibility between writer and reader and should not be left to the compositor. An author, when he finds himself writing a longish formula, should indicate a convenient point of fracture in case of need.

— CHAUNDY, BARRETT, and BATEY, *The Printing of Mathematics* (1954)

Некоторые авторы тактично выделяют формулы, некоторые даже слишком длинные и сложные уравнения вставляют в текст, в то время как другие выделяют каждое уравнение в статье. Преобладает тенденция перевыделения, а не недовыделения, поэтому редактор может сократить (и даже улучшить) внешний вид, вставляя выделенный материал в текст. ... С другой стороны, бывают случаи, когда редактор должен предложить выделить сложное выражение, которое располагалось в тексте, особенно если это приводило к плохому разбиению текстовой строки.

Some authors use display with discretion, some run even extremely long, complicated equations into the text, while others tend to display every equation in the paper. The tendency to overdisplay is probably more predominant than the tendency to underdisplay; for this reason it is possible for the copy editor to shorten (and even improve) papers by running displayed material into text. ... On the other hand, there are occasions when the copy editor needs to suggest the display of complicated expressions that have been run into text, particularly when it would involve a bad break at the end of a text line.

— ELLEN SWANSON, *Mathematics into Type* (1971)

20

Определения (или Макрокоманды)

Часто при наборе математических формул можно сэкономить время, обозначая конструкции, которые часто встречаются, командами. Например, если часто используется вектор $(x_1,\ldots,x_n)$, можно задать

```
\def\xvec{(x_1,\ldots,x_n)}
```

и `\xvec` впредь будет сокращением для `(x_1,\ldots,x_n)`. Тогда такая сложная выделенная формула

$$\sum_{(x_1,\ldots,x_n)\ne(0,\ldots,0)} \bigl(f(x_1,\ldots,x_n)+g(x_1,\ldots,x_n)\bigr)$$

может быть введена просто как

```
$$\sum_{\xvec\ne(0,\ldots,0)} \bigl(f\xvec+g\xvec\bigr)$$
```

вместо скучной длинной формы. Определив команду `\xvec`, вы не только сокращаете число нажатий клавишей, которые надо сделать, а также уменьшаете шансы на внесение опечаток и недоразумений.

Конечно, обычно не стоит делать определения только для того, чтобы ускорить ввод одной формулы. Это вам ничего не даст, потому что пройдет время, пока вы решите, делать или не делать определение, и пока введете само определение. Реальная выгода появляется тогда, когда некоторые группы символов на протяжении рукописи используются десятки раз. Умный наборщик, прежде чем печатать что–либо, просмотрит весь документ, чтобы прочувствовать, какого сорта проблемы возникнут и какого сорта определения будут полезны. Например, глава 16 рекомендует, чтобы команда `\Ahat` была определена в начале рукописей, которые часто используют символ $\hat{A}$.

Сокращения типа `\xvec` оказываются полезны во многих приложениях и получили известность как *макрокоманды* из–за их мощности: одна маленькая макрокоманда может представлять огромное количество материала, так что она обладает некоторого рода макроскопическим эффектом. Программные системы типа TEX'а, которые имеют макроопределения, должны уметь *раскрывать* макрокоманды пользователей. Например, `\xvec` раскрывается в `$(x_1,\ldots,x_n)$`, а `\ldots` оказывается макрокомандой, которая раскрывается в `\mathinner{\ldotp\ldotp\ldotp}`. Таким образом, `\xvec` в действительности является сокращением для

```
$(x_1,\mathinner{\ldotp\ldotp\ldotp},x_n)$.
```

(Раскрытие на этом прекращается, поскольку, `\mathinner` — это примитивная команда TEX'а, а `\ldotp` определена при помощи `\mathchardef`; таким образом, `\mathinner` и `\ldotp` — это не макрокоманды.)

Пользователи TEX'а обычно строят свою собственную библиотеку макроопределений в зависимости от того, что им нужно делать в различных документах. Например, принято иметь файл, называемый `macros.tex`, который содержит определения для ваших любимых специальных команд,

возможно, вместе с командами, которые загружают ваши любимые специальные шрифты, и так далее. Если вы начинаете работу с команды

```
\input macros
```

то TEX прочитывает все эти определения и ограждает вас от ошибок при их перепечатывании. Конечно, поскольку память TEX'а ограничена, а для чтения файла требуется дополнительное время, вы не должны в `macros.tex` помещать слишком много определений. Большой набор макроопределений (например, множество определений в приложении B) называется *форматом* (например, формат начального TEX'а). У TEX'а есть специальный способ максимально быстро ввести формат в предположении, что формат меняется не слишком часто.

Команды `\xvec` и `\Ahat` применяются в математических формулах, но макроопределения удобны не только в математике. Например, когда TEX используется для деловой переписки, можно иметь макроопределение `\yours`, которое обозначает "Искренне Ваш, А. В. Тор". Если вы часто пишите стандартные письма, вам полезны макроопределения, которые создают целые предложения, абзацы или группы абзацев. Во Внутреннем налоговом управлении (Internal Revenue Service) можно, например, иметь следующие два макроопределения:

```
\def\badcheck{Налагается штраф, поскольку ваш чек
  не оплачен вашим банком.\par}
\def\cheater{Добавлен штраф в 50\% от нижней ставки за
  мошенничество.\par}
```

Такие простые макроопределения начинаются с `\def`, затем следует имя команды, например `\badcheck`, а затем текст замены, заключенный в `{` и `}`. Фигурные скобки в этом случае не служат группированию, они просто указывают протяженность текста замены в определении. Можно определить макрокоманду, которая включает действующие фигурные скобки в тексте замены, если только эти скобки правильно сочетаются друг с другом. Например, `\def\xbold{{\bf x}}` делает `\xbold` сокращением для `{\bf x}`.

▸ **Упражнение 20.1**

Напишите макроопределение `\punishment`, которое печатает 100 строк сообщений "Я не должен болтать в классе." [*Намек:* сначала напишите макроопределение `\mustnt`, которое печатает сообщение один раз, затем напишите макроопределение `\five`, которое печатает его пять раз.]

▸ **Упражнение 20.2**

Во что раскрывается `\puzzle`, заданное следующими определениями?

```
\def\a{\b}
\def\b{A\def\a{B\def\a{C\def\a{\b}}}}
\def\puzzle{\a\a\a\a\a}
```

Поскольку вы разобрались в простых макроопределениях, которые иллюстрировались выше, вы, вероятно подумываете: "Парень, было бы прекрасно, если бы я мог иметь такое макроопределение, в котором некоторый текст можно было бы заменять! Мне бы хотелось вставлять различные штуки в середину этого текста." Прекрасно, у TEX'а для вас хорошая новость: есть команды с *параметрами*, и можно использовать *аргументы*, которые будут заменять параметры.

Например, давайте снова рассмотрим `\xvec`. Предположим, что вы используете не только $(x_1,\ldots,x_n)$, но и $(y_1,\ldots,y_n)$ и другие аналогичные штуки. Тогда вы можете ввести

```
\def\row#1{(#1_1,\ldots,#1_n)}
```

после чего `\row x` будет давать $(x_1,\ldots,x_n)$ а `\row y` даст $(y_1,\ldots,y_n)$. Символ `#1` обозначает первый параметр макроопределения, и когда вы говорите `\row x`, `x` будет так называемым аргументом, который вставляется в текст замены вместо `#1`. В этом случае аргумент состоит из одной буквы `x`. Можно также сказать `\row\alpha`, и в этом случае аргументом будет команда `\alpha`, а результатом — $(\alpha_1,\ldots,\alpha_n)$. Если надо, чтобы аргумент состоял более чем из одного символа или команды, можно заключить его в фигурные скобки. Например, `\row{x'}` дает $(x'_1,\ldots,x'_n)$. Аргументом в этом случае является `x'` (без фигурных скобок). Между прочим, если вы говорите `\row{{x'}}`, то получаете $(x'_1,\ldots,x'_n)$. Причина в том, что когда выбирается аргумент, одна пара фигурных скобок отбрасывается, и в математической моде в соответствии с правилами главы 16 из `({x'}_1,\ldots,{x'}_n)` получается $(x'_1,\ldots,x'_n)$

▸ **Упражнение 20.3**
Продолжим этот пример. Какой результат даст `$\row{\bf x}$`?

Запись `#1` предполагает, что можно иметь более одного параметра, и, конечно же, так оно и есть. Вы можете, например, написать

```
\def\row#1#2{(#1_1,\ldots,#1_#2)}
```

после чего `\row xn` будет правильным сокращением для $(x_1,\ldots,x_n)$. Можно иметь до девяти параметров от `#1` до `#9`, и когда вы используете их, то должны перечислить их по порядку. Например, нельзя в определении использовать `#5`, если предыдущий параметр этого определения не был `#4`. (Это ограничение применяется только при начальном задании параметров, перед тем, как начинается текст замены; в самом тексте замены заданные параметры могут быть использованы любое число раз в любом порядке.)

Команда может одновременно иметь только одно определение, так что второе определение `\row` перекроет первое, если они оба появятся в одном и том же документе. Когда TEX встречает макроопределение, которое хочет раскрыть, он использует самое свежее определение. Однако, определения являются локальными в группе, которая их содержит. Старые определения будут обычным образом восстановлены, когда группа закончится.

Предупреждение: когда вы определяете макрокоманду с простыми параметрами, как в этих примерах, будьте внимательны и не ставьте пробел

перед {, которая открывает текст замены. Например, `\def\row #1 #2 {...}` даст не тот же самый результат, что `\def\row#1#2{...}`, поскольку пробелы после `#1` и `#2` указывают TEX'у искать аргументы, за которыми следуют пробелы. (Аргументы могут быть "ограничены", как объяснено ниже, достаточно общим способом.) Но пробел после `\row`, как обычно, является не обязательным, поскольку TEX всегда игнорирует пробелы после команд. После того, как вы сказали `\def\row#1#2{...}`, вам разрешается поставить пробелы между аргументами (например, `\row x n`), поскольку TEX не использует одиночные пробелы, как аргументы без ограничителей.

Следующее упражнение особенно рекомендуется тем, кто хочет научиться писать макроопределения. Даже если вы уже приобрели дурную привычку лишь просматривать другие упражнения, это упражнение вы должны попытаться выполнить.

▸ **Упражнение 20.4**

Продолжая упражнение 20.1, напишите макроопределение "обобщенного наказания", которое имеет два параметра, так чтобы `\punishment{бегать}{по коридору}` производил 100 абзацев, которые говорят "Я не должен бегать по коридору."

TEX позволяет определять макрокоманды, параметры которых ограничены обычным образом, поэтому не всегда надо заключать аргументы в скобки. Например,

```
\def\cs #1. #2\par{...}
```

определяет команду `\cs` с двумя параметрами, а ее два аргумента определяются следующим образом: `#1` состоит из всех элементов между `\cs` и `.␣` (точка с пробелом), а `#2` состоит из всех элементов между этим `.␣` и следующим элементом `\par`. (`\par` может быть задан явно или генерирован пустой строкой, как объяснялось в главе 8.) Например, когда TEX раскрывает

```
\cs Вы должны \$5.00. Заплатите их.\par
```

то первым аргументом будет "`Вы должны \$5.00`", а вторым — "`Заплатите их.`". В этом примере точка в `\$5.00` не заканчивает `#1`, поскольку TEX читает аргумент до тех пор, пока не найдет точку со следующим за ней пробелом.

Кроме того, аргумент не заканчивается, если его ограничитель заключен в фигурные скобки. Например, в

```
\def\cs #1.#2\par{...}
```

первый аргумент теперь ограничен одной точкой, поэтому, если бы `\cs` вызывалось, как раньше, параметр `#1` был бы таким: "`Вы должны \$5`", а параметр `#2` таким: "`00. Заплатите их.`". Но

```
\cs Вы должны {\$5.00}. Заплатите их.\par
```

благополучно прихватывает первую точку, делая ее частью аргумента `#1`, который тогда становиться таким: "`Вы должны {\$5.00}`".


Если вы разрабатываете формат для математических документов, вам, вероятно, захочется иметь макроопределения для формулировок теорем, определений, лемм, следствий и тому подобных вещей. Например, может быть вы захотите напечатать такое утверждение:

Теорема 1. *TEX имеет макроопределения.*

и для этого ввести:

```
\proclaim Теорема 1. \TeX\ имеет  макроопределения.\par
```

Начальный TEX содержит макрокоманду `\proclaim`, которая делает именно это, и ее определение таково:

```
\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2}\par\medbreak}
```

так что аргументы разграничены точно так же, как в нашем первом примере `\cs`. Текст замены здесь использует `\medbreak`, чтобы отделить формулировку от того, что находится до и после нее; заголовок задан жирным шрифтом, а сам текст — наклонным. (В действительности в приложении B приведено несколько другое определение `\proclaim`, там `\medbreak` модифицирован так, что не рекомендуется делать разрыв страницы сразу за утверждением теоремы. Следовательно, короткая теорема скорее всего появится наверху, а не внизу страницы.)


Изменяя макроопределение `\proclaim`, можно изменить формат всех формулировок теорем в документе, не меняя текст самой рукописи. Например, можно получить и такое:

ТЕОРЕМА 1: *TEX имеет макроопределения.*

сделав простые изменения в тексте замены `\proclaim`, если у вас есть шрифт "капитель". С помощью TEX'а можно создавать языки высокого уровня для такого способа набора, при котором команды, которые пользователь вводит, чаще являются макроопределениями, чем примитивами TEX'а. Идеалом является возможность описывать важные классы документов в терминах их компонентов, не упоминая реальные шрифты, размеры в пунктах или подробности распределения пробелов. Тогда независимый от стиля документ может быть затем получен во многих различных стилях.


Теперь после того, как мы разобрали несколько примеров, давайте рассмотрим точные правила, которым подчиняются макроопределения TEX'а. Определения имеют общий вид

`\def`⟨команда⟩⟨текст параметров⟩`{`⟨текст замены⟩`}`

где текст параметров не содержит фигурных скобок, а все `{` и `}` в тексте замены правильно вложены. Символ `#` имеет специальное значение: в тексте параметров за первым `#` должен следовать `1`, за следующим — `2`, и так далее; разрешено до девяти `#`. В тексте замены за каждым `#` должна следовать цифра, которая появлялась после `#` в тексте параметров, или другой `#`. Последний случай, когда макроопределение раскрывается, обозначает один элемент `#`; первый случай обозначает вставку соответствующего аргумента.

Например, давайте рассмотрим взятое наугад определение, которое не делает ничего полезного, а только демонстрирует правила TEX'а. Определение

```
\def\cs AB#1#2C$#3\$ {#3{ab#1}#1 c##\x #2}
```

говорит, что команда `\cs` имеет текст параметров, состоящий из девяти элементов

$$\texttt{A}_{11},\ \texttt{B}_{11},\ \texttt{\#1},\ \texttt{\#2},\ \texttt{C}_{11},\ \texttt{\$}_{3},\ \texttt{\#3},\ \boxed{\$},\ \texttt{␣}_{10}$$

(предполагая номера категорий начального TEX'а), и текст замены, состоящий из двенадцати элементов

$$\texttt{\#3},\ \texttt{\{}_{1},\ \texttt{a}_{11},\ \texttt{b}_{11},\ \texttt{\#1},\ \texttt{\}}_{2},\ \texttt{\#1},\ \texttt{␣}_{10},\ \texttt{c}_{11},\ \texttt{\#}_{6},\ \boxed{\texttt{x}},\ \texttt{\#2}.$$

Впредь, когда TEX прочитывает команду `\cs`, он ожидает, что следующие два элемента будут $\texttt{A}_{11}$ и $\texttt{B}_{11}$ (иначе вы получите сообщение об ошибке: "`Использование \cs не соответствует его определению`"), затем следует аргумент `#1`, затем аргумент `#2`, затем $\texttt{C}_{11}$, затем $\texttt{\$}_{3}$, затем аргумент `#3`, затем `\$` и, наконец, элемент-пробел. Принято использовать слово "аргумент" для обозначения строки элементов, которая замещает параметр. Параметры появляются в определении, а аргументы появляются, когда это определение используется. (Для этих правил мы расширяем определение элемента, данное в главе 7: в дополнение к командам и парам вида (код символа, номер категории), TEX также распознает "параметрические элементы", обозначенные здесь от `#1` до `#9`. Параметрические элементы могут появляться только в списках элементов для макроопределений.)

Вы спросите, как TEX определяет, где оканчивается аргумент. Ответ: есть два случая. Первый — за *ограниченным параметром* в тексте параметров следует один или более непараметрический элемент до конца текста параметров или до следующего параметрического элемента. В этом случае соответствующий аргумент — это наименьшая (возможно, пустая) последовательность элементов с правильно вложенными `{...}`-группами, за которыми во входном потоке следует этот список непараметрических элементов. (Номера категорий и коды символов должны соответствовать друг другу, а имена команд должны быть теми же самыми.) Второй — за *неограниченными параметрами* в тексте параметров либо сразу следует параметрический элемент, либо этот параметр встречается в самом конце текста параметров. Тогда соответствующим аргументом является следующий непробельный элемент, за исключением случая, когда этим элементом является `{`, а аргументом будет вся следующая группа `{...}`. В обоих случаях, если аргумент, найденный таким способом, имеет форму "`{`⟨вложенные элементы⟩`}`", где ⟨вложенные элементы⟩ обозначают любую последовательность элементов, которые правильно вложены с учетом фигурных скобок, то самые внешние скобки, заключающие аргумент, убираются и остаются только ⟨вложенные элементы⟩. Например, продолжим рассмотрение `\cs`, определенного выше, и предположим, что последующий текст содержит:

```
\cs AB {\Look}C${And\$ }{Look}\$ 5.
```

Аргументом `#1` будет элемент `Look`, поскольку `#1` — неограниченный параметр (за ним в определении непосредственно следует `#2`); в этом случае TEX игнорирует

пробел после `B` и отбрасывает фигурные скобки в `{\Look}`. Аргумент `#2` будет пустым, поскольку сразу же идет `C$`. А аргументом `#3` будут тринадцать элементов, соответствующие тексту `{And\$␣}{Look}`, поскольку за `#3` следует `\$␣` и поскольку первый раз `\$␣` появляется внутри скобок. Несмотря на то, что аргумент `#3` начинается с левой фигурной скобки и оканчивается правой фигурной скобкой, фигурные скобки не убираются, поскольку это оставило бы невложенные элементы "`And\$ }{Look`". Тогда результат после подстановки в тексте замены будет такой: TEX будет читать список элементов

```
{And\$ }{Look}{ab\Look}\Look␣c#\x5.
```

Пробел `␣` здесь будет частью результирующего списка элементов, несмотря на то, что следует за командой `\Look`, потому что пробелы после элементов-команд убираются, только когда TEX в первый раз преобразует входные строки в списки элементов, как это описано в главе 8.

▸ **Упражнение 20.5**
Пример определения `\cs` включает в свой текст замены `##`, но способ, которым `##` использован в этом примере, достаточно глупый. Приведите пример определения, когда `##` служит полезной цели.

Допускается специальное расширение этих правил: если самым последним символом текста параметров является `#`, так что за этим `#` сразу же следует `{`, TEX ведет себя, как если бы `{` была вставлена на правом конце как текста параметров, так и текста замены. Например, если вы говорите `\def\a#1#{\hbox to #1}`, то следующий текст `\a3pt{x}` будет раскрываться в `\hbox to 3pt{x}`, поскольку аргумент `\a` ограничен левой фигурной скобкой.

Элементы, которые предшествуют правому параметрическому элементу в тексте параметров определения должны следовать за командой; по существу, они становятся частью имени команды. Например, автор мог сказать

```
\def\TeX/{...}
```

вместо того, чтобы определять `\TeX` без косой черты. Тогда каждый раз при создании эмблемы TEX было бы необходимо вводить `\TeX/`, но новое определение имело бы то преимущество, что пробелы после "`\TeX/`" *не* игнорируются. Эту идею можно использовать, чтобы определить макрокоманды, которые так используются в тексте, что пользователи могут не беспокоиться об исчезновении пробелов.

▸ **Упражнение 20.6**
Определите такую команду `\a`, что `\a{...}` раскрывается в `\b{...}`, и что TEX выдает сообщение об ошибке, если за `\a` сразу не следует левая фигурная скобка.

Сложные макроопределения, когда вы их впервые определяете, имеют привычку вести себя не так, как вы ожидаете, хотя правила TEX'а и не особенно сложны. Если у вас недоразумения с пониманием того, почему какая-то `\def` работает не так, как она, по вашему мнению, должна работать, то можно получить помощь. Вы можете установить `\tracingmacros=1`, после чего TEX, когда раскрывает макроопределение и прочитывает аргументы, будет писать кое-что в ваш протокольный файл. Например, если `\tracingmacros` положительно в то

время, когда TEX обрабатывает приведенный выше пример `\cs`, он запишет в протокол следующие четыре строки:

```
\cs AB#1#2C$#3\$ ->#3{ab#1}#1 c##\x #2
#1<-\Look
#2<-
#3<-{And\$ }{look}
```


Во всех правилах, сформулированных выше, вместо `{` , `}` и `#` могут быть любые символы, категории которых в списке символов, когда TEX читает макроопределение, равны, соответственно, 1,2 и 6; не следует считать неприкосновенными конкретные символы, которые начальный TEX использует для группирования и параметров. Можно даже одновременно использовать несколько различных символов с этими номерами категорий.


▸ **Упражнение 20.7**
Предположим, что `[`, `]` и `!` имеют, соответственно, номера категорий 1, 2, и 6, так же, как `{`, `}` и `#`. Разберитесь, если можете, что означает следующее определение:

```
\def\!!1#2![{!#]#!!2}
```

Какой получится список элементов, когда будет раскрыто "`\! x{[y]][z}`"?


На практике все мы делаем опечатки. И одна из наиболее общих опечаток — это забыть `}` или вставить лишний `{` где-нибудь в аргументах макрокоманды. Если бы TEX в таком случае слепо следовал правилам, он поглощал бы все больше и больше элементов в надежде найти конец аргумента. Но ошибочный аргумент бесконечен, как и большинство аргументов в реальной жизни (вздох), так что TEX должен был бы продолжать это до конца файла или, что более вероятно, пока элементы полностью не заполнят память компьютера. В любом случае простая опечатка испортила бы обработку, и пользователь был бы вынужден повторить запуск. Поэтому у TEX'а есть еще одно правило, призванное ограничить такие ошибки абзацем, в котором они встретились: *элементу* `\par` *не позволяется быть частью аргумента*, если вы только не указали явно TEX'у, что с `\par` все в порядке. Как только TEX попытается включить `\par` в аргумент, он прервет макроопределение и сообщит, что обнаружен "убегающий аргумент".


Если же вам хочется иметь команду, у которой разрешены аргументы с элементами `\par`, вы можете определить "длинную" макрокоманду, указав `\long` прямо перед `\def`. Например, макрокоманда `\bold`, определенная как

```
\long\def\bold#1{{\bf#1}}
```

может задавать жирный шрифт для абзацев. (Однако, использовать такую макрокоманду для печати жирным шрифтом не очень хорошо. Лучше было бы, например, сказать

```
\def\beginbold{\begingroup\bf}
\def\endbold{\endgroup}
```

поскольку это не занимает память TEX'а длинным аргументом.)

Тем не менее, механизм, запрещающий `\par`, не вылавливает все очевидные ошибки, вызванные пропуском фигурных скобок: вы могли забыть `}` в конце `\def`, и возникла бы та же самая проблема. В этом случае труднее ограничить ошибку, поскольку `\par` полезен в тексте замены, и нам не хотелось бы запрещать его там, так что у TEX'а есть другой механизм. Когда макроопределению предшествует `\outer`, соответствующей команде будет не разрешено появляться в том месте, где происходит скоростное поглощение элементов. Макрокоманда `\outer` не может появляться ни в аргументе (даже когда разрешено `\par`) ни в тексте параметров, ни в тексте замены определения, ни в преамбуле к выравниванию, ни в условном тексте, который пропускается. Если `\outer` все же появляется в таких местах, TEX прекращает свои действия и сообщает либо о ситуации "убегающего аргумента", либо о "незавершенном" условии. Конец входного файла так же рассматривается в этом смысле как `\outer`. Например, файл не должен оканчиваться в середине определения. Если вы разрабатываете формат для того, чтобы его использовали другие пользователи, то можете помочь им обнаруживать ошибки до того, как причинен большой ущерб. Для этого используйте `\outer` со всеми командами, которые должны появляться только "тайно" внутри документа. Например, приложение B определяет `\proclaim` как `\outer`, поскольку пользователь не должен формулировать теорему как часть определения, аргумента или преамбулы.

Теперь мы видели, что `\def` может предваряться `\long` или `\outer`, а им также может предшествовать `\global`, если предполагается, что определение выходит за границы группы. Эти три приставки могут прилагаться к `\def` в любом порядке, и даже могут появляться более одного раза. TEX также имеет примитив `\gdef`, который эквивалентен `\global\def`. Так, например,

```
\long\outer\global\long\def
```

означает то же самое, что `\outer\long\gdef`.

До сих пор в этом руководстве мы встречались с несколькими способами присваивания значений командам. Например,

`\font\cs=`⟨внешнее имя шрифта⟩	делает `\cs` идентификатором шрифта;
`\chardef\cs=`⟨число⟩	делает `\cs` символьным кодом;
`\countdef\cs=`⟨число⟩	делает `\cs` `\count`-регистром;
`\def\cs...{...}`	делает `\cs` макрокомандой.

Сейчас покажем еще одну важную команду этого типа:

`\let\cs=`⟨элемент⟩	придает `\cs` текущее значение элемента.

Если ⟨элемент⟩— это другая команда, то `\cs` приобретет то же самое значение, что и эта команда. Например, если вы говорите `\let\a=\def`, то затем, чтобы определить макрокоманду `\b`, можно сказать `\a\b...{...}`, потому что `\a` действует, как примитивная команда TEX'а `\def`. А если вы говорите

```
\let\a=\b \let\b=\c \let\c=\a
```

то обмениваете предыдущие значения `\b` и `\c`. А если говорите

```
\outer\def\a#1.{#1:}
\let\b=\a
```

эффект будет в точности такой же, как при `\outer\def\b#1.{#1:} \let\a=\b`.


Если ⟨элемент⟩ в `\let` — это один символ, т.е. пара (символьный код, номер категории), то команда будет работать в какой-то мере, как этот символ, но с некоторыми различиями. Например, после `\let\zero=0` вы не можете использовать `\zero` в числовой константе, поскольку TEX требует, чтобы элементы в числовой константе после макроопределения были десятичными цифрами; `\zero` не является макроопределением, поэтому не раскрывается. Однако, как мы увидим позднее, такое использование `\let` имеет смысл.


▸ **Упражнение 20.8**
Имеется ли существенное различие между `\let\a=\b` и `\def\a{\b}`?


▸ **Упражнение 20.9**
Поэкспериментируйте с TEX'ом, чтобы ответить на следующие вопросы: (a) Если команда `\par` переопределена (например, `\def\par{\endgroup\par}`), то ей все еще запрещено появляться в аргументе? (b) Если вы говорите `\let\xpar=\par`, то `\xpar` так же запрещается присутствовать в аргументе?


TEX также разрешает конструкцию `\futurelet\cs`⟨элемент$_1$⟩⟨элемент$_2$⟩, которая действует как `\let\cs =` ⟨элемент$_2$⟩⟨элемент$_1$⟩⟨элемент$_2$⟩. Идея здесь в том, что вы, например, можете сказать, `\futurelet\a\b` в конце текста замены макроопределения. TEX будет устанавливать `\a` равным элементу, следующим за макроопределением, после чего `\b` будет раскрываться. Команда `\b` может продолжить обработку и проверить содержание `\a`, чтобы посмотреть, что будет дальше.


Следующее, что каждому хочется после того, как получены макроопределения с параметрами, это возможность писать макроопределения, которые меняют свое поведение в зависимости от текущих условий. Для этой цели TEX предусматривает множество примитивных команд. Общий вид таких "условных текстов" следующий:

`\if`⟨условие⟩⟨правильный текст⟩`\else`⟨ложный текст⟩`\fi`

где ⟨правильный текст⟩ пропускается, если ⟨условие⟩ не верно, а ⟨ложный текст⟩ пропускается, если ⟨условие⟩ не ложно. Если ⟨ложный текст⟩ пустой, вы можете опустить `\else`. Часть "`\if`⟨условие⟩" в этой конструкции начинается с команды, первые две буквы которой "`if`"; например,

```
\ifodd\count0 \rightpage \else\leftpage \fi
```

задает условие, которое истинно, когда целый регистр TEX'а `\count0` нечетный. Поскольку TEX обычно содержит в `\count0` номер текущей страницы, то в этом примере на страницах с нечетным номером макроопределение будет раскрываться в `\rightpage`, а на страницах с четным номером — в `\leftpage`. Условные команды всегда завершаются конечным `\fi`.


Условия в основном предназначены для опытных пользователей, которые хотят определять макрокоманды высокого уровня, поэтому оставшиеся абзацы этой главы помечены "двойными знаками опасного поворота". Не чувствуйте себя виноватыми, если перескочите прямо на главу 21. Другими словами, представьте, что прямо здесь, в руководстве сказано `\ifexperienced`, а `\fi` следует в конце этой главы.

Перед тем, как обсуждать репертуар команд `\if...`, давайте рассмотрим еще один пример, который прояснит общие идеи. Предположим, что `\count`-регистр `\balance` содержит сумму, которую кто-то заплатил сверх своего подоходного налога. Эта сумма задана в пенни, и может быть положительной, отрицательной или нулевой. Нашей целью сейчас будет написать макроопределение, которое генерирует подходящее утверждение для Внутреннего налогового управления, чтобы включить его как часть письма этому лицу, составленного в зависимости от суммы баланса. Письмо для положительного баланса будет совсем не такое, как для отрицательного баланса, поэтому мы будем использовать способность TEX'а действовать условно:

```
\def\statement{\ifnum\balance=0 \fullypaid
  \else\ifnum\balance>0 \overpaid
       \else\underpaid
       \fi
  \fi}
```

Здесь `\ifnum` — это условная команда, которая сравнивает два числа; макроопределение `\statement` превращается в `\fullypaid`, если баланс нулевой и т.д.

Чрезвычайно важно отметить в этой конструкции пробелы после нулей. Если бы в примере было сказано

```
...=0\fullypaid...
```

то TEX начал бы раскрывать `\fullypaid` перед тем, как узнал значение константы `0`, поскольку `\fullypaid` могла начинаться с `1` или чего-нибудь другого, что изменило бы число. (В конце концов, `01` в глазах TEX'а—это абсолютно допустимое ⟨число⟩) В нашем конкретном случае программа еще бы работала, поскольку, как мы сейчас увидим, `\fullypaid` начинается с буквы `B`; таким образом, единственной проблемой, связанной с пропуском пробела, будет то, что TEX будет работать медленнее, поскольку будет перескакивать через все раскрытия `\fullypaid` вместо того, чтобы пропустить только `\fullypaid`, как один нераскрытый элемент. Но в других случаях такой пропущенный пробел мог указать TEX'у раскрыть макроопределение, когда вы вовсе не хотите этого, а такие аномалии могут привести к тонким и запутанным ошибкам. Лучше *всегда ставить пробел после числовой константы*; такой пробел говорит TEX'у, что константа завершена, и к тому же этот пробел никогда "не проскочит" в выходной результат. Действительно, когда у вас нет пробела после константы, TEX должен выполнить больший объем работы, поскольку каждая константа продолжается до тех пор, пока не будет прочитана не-цифра. Если эта не-цифра не пробел, TEX берет элемент, который у него был, и возвращает его, готовый прочесть опять. (С другой стороны, автор часто опускает пробелы, когда за константой непосредственно следует какой-нибудь другой символ, поскольку дополнительные пробелы в файле выглядят странно, а эстетика не менее важна, чем эффективность.)

Упражнение 20.10

Продолжим пример с Налоговым управлением. Предположим, что у нас

`\fullypaid` и `\underpaid` определены следующим образом:

```
\def\fullypaid{Ваши налоги полностью уплачены --- благодарим вас.}
\def\underpaid{{\count0=-\balance
  \ifnum\count0<100
    Вы должны \dollaramount, но вам не надо платить их, поскольку
    наша политика --- игнорировать суммы меньше, чем \$1.00.
  \else Пожалуйста, внесите \dollaramount\ в течении десяти дней,
    или с вас будут удержаны дополнительные проценты.\fi}}
```

Напишите макрокоманду `\overpaid`, предполагая, что `\dollaramount` — это макроопределение, которое генерирует содержимое `\count0` в долларах и центах. Ваша макрокоманда должна говорить, что чек будет послан в отдельном конверте, если только сумма не меньше, чем \$1.00, в этом случае клиент должен специально запросить чек.

▸ **Упражнение 20.11**
Напишите макроопределение `\dollaramount`, чтобы завершить макроопределение `\statement` для Налогового управления.

Теперь давайте сделаем полный обзор условных команд TEX'а. Некоторые из них содержат детали, которые в этом руководстве еще не описывались.

- `\ifnum`⟨число$_1$⟩⟨отношение⟩⟨число$_2$⟩ (сравнение двух чисел)

Здесь ⟨отношение⟩ должно быть либо `<`$_{12}$, либо `=`$_{12}$, либо `>`$_{12}$. Два целых числа сравниваются обычным образом и результат будет, соответственно, истинным либо ложным.

- `\ifdim`⟨размер$_1$⟩⟨отношение⟩⟨размер$_2$⟩ (сравнение двух размеров)

Это похоже на `\ifnum`, но сравнивает значения ⟨размер⟩ . Например, для проверки, не превышает ли значение `\hsize` 100 pt, можно сказать `\ifdim\hsize>100pt`.

- `\ifodd`⟨число⟩ (проверка нечетности)

Условие истинно, если ⟨число⟩ нечетно, и ложно, если четно.

- `\ifvmode` (проверка вертикальной моды)

Истинно, если TEX находится в вертикальной или внутренней вертикальной моде (см. главу 13).

- `\ifhmode` (проверка горизонтальной моды)

Истинно, если TEX находится в горизонтальной моде или частной горизонтальной моде (см. главу 13).

- `\ifmmode` (проверка математической моды)

Истинно, если TEX находится в математической моде или выделенной математической моде (см. главу 13).

- `\ifinner` (проверка внутренней моды)

Истинно, если TEX находится во внутренней вертикальной , частной горизонтальной или (невыделенной) математической моде (см. главу 13).

■ `\if`⟨элемент$_1$⟩⟨элемент$_2$⟩ (проверка, согласуются ли символьные коды)

TEX будет раскрывать макроопределения, следующие за `\if`, до тех пор, пока не найдутся два нераскрываемых элемента. Если любой из этих элементов является командой, TEX рассматривает его как имеющего символьный код 256 и номер категорий 16, исключая случай, когда текущий эквивалент этой команды `\let`-равен неактивному символьному элементу. В этом случае каждый элемент задает пару (символьный код, номер категории). Условие истинно, если равны символьные коды, независимо от номеров категорий. Например, после `\def\a{*}`, `\let\b=*` и `\def\c{/}`, проверки `\if*\a` и `\if\a\b` будут истинными, а `\if\a\c` —ложной. Также `\if\a\par` будет ложной, а `\if\par\let` — истинной.

■ `\ifcat`⟨элемент$_1$⟩⟨элемент$_2$⟩ (проверка согласования номера категорий)

Это похоже на `\if`, но проверяются номера категорий, а не символов. Активные символы имеют категорию 13, но когда вы рассматриваете такие символы при помощи `\if` или `\ifcat`, надо сказать `\noexpand<активный символ>`, чтобы подавить раскрытие. Например, после

```
\catcode`[=13 \catcode`]=13 \def[{*}
```

проверки "`\ifcat\noexpand[\noexpand]`" и "`\ifcat[*`" будут истинными, а проверка "`\ifcat\noexpand[*`" — ложной.

■ `\ifx`⟨элемент$_1$⟩⟨элемент$_2$⟩ (проверка согласования элементов)

В этом случае TEX *не* раскрывает команды, когда смотрит на два элемента. Условие истинно, когда (a) два элемента не являются макроопределениями и оба представлены одинаковыми парами (символьный код, номер категории), одинаковым примитивом TEX'a или одним и тем же `\font`, или `\chardef`, или `\countdef` и т.д., или (b) оба элемента являются макроопределениями, оба имеют один и тот же статус по отношению к `\long` и `\outer` и оба имеют одни и те же параметры и раскрытия "высшего уровня". Например, после `\def\a{\c} \def\b{\d} \def\c{\e} \def\d{\e} \def\e{A}` проверка `\ifx` обнаружит, что `\c` и `\d` равны, но не равны ни `\a` и `\b`, ни `\d` и `\e`, ни другие какие-либо комбинации `\a`, `\b`, `\c`, `\d`, `\e`.

■ `\ifvoid`⟨число⟩, `\ifhbox`⟨число⟩, `\ifvbox`⟨число⟩ (проверка регистра бокса)

Здесь ⟨число⟩ должно быть между 0 и 255. Условие истинно, если соответственно, `\box` пустой, содержит h-бокс или v-бокс. (см. главу 15).

■ `\ifeof`⟨число⟩ (проверка на конец файла)

Здесь ⟨число⟩ должно быть между 0 и 15. Условие истинно, за исключением случая, когда соответствующий входной поток открыт и не полностью прочитан. (См. ниже команду `\openin`.)

■ `\iftrue`, `\iffalse` (всегда истинны или всегда ложны)

Эти условия имеют заранее заданный результат. Но несмотря на это, как объясняется ниже, они иногда оказываются полезными.

Наконец, есть еще одна условная конструкция, которая несколько отличается от остальных, поскольку она способна сделать разветвление по нескольким путям:

■ `\ifcase`⟨число⟩⟨текст для случая 0⟩`\or`⟨текст для случая 1⟩`\or` ⋯

`\or`⟨текст для случая n⟩`\else`⟨текст для других случаев ⟩`\fi`

Здесь $n+1$ случаев разделены через n '`\or`', где n может быть любым неотрицательным числом. ⟨Число⟩ выбирает текст, который будет использован. И опять, часть, начинающаяся с `\else`, не обязательна, если вы не хотите указать какой-либо текст для случая, когда ⟨число⟩ отрицательно либо больше n.

Упражнение 20.12
Создайте макрокоманду `\category`, которая символически печатает текущий номер категории символа, задавая для этого символа односимвольную команду. Например, если лействуют номера категорий начального TEX'a, `\category\\` должна раскрываться в сигнальный символ, а `\category\a` — в "`буква`".

Упражнение 20.13
Проверьте на следующих вопросах, как вы поняли некоторые граничные ситуации: после цепочки определений `\def\a{} \def\b{**} \def\c{True}`, какие из команд дают значение "истина"? (a) `\if\a\b`; (b) `\ifcat\a\b`; (c) `\ifx\a\b`; (d) `\if\c`; (e) `\ifcat\c`; (f) `\ifx\ifx\ifx`. (g) `\if\ifx\a\b\c\else\if\a\b\c\fi\fi`.

Заметим, что все команды для сравнений начинаются с `\if...` и имеют парную `\fi`. Это условие — что `\if...` спаривается с `\fi` — помогает легче увидеть вложенные условия внутри вашей программы. Вложение `\if...\fi` независимо от вложения `{...}`; так, вы можете начинать или оканчивать группу в середине условия и начинать или оканчивать условия в середине группы. Широкий опыт работы с макроопределениями показывает, что такая независимость важна в приложениях, но если вы невнимательны, она также может привести к путанице.

Иногда желательно передать информацию из одной макрокоманды в другую. Для этого есть несколько способов: передавать ее как аргумент, помещать в регистр или определить команду, которая содержит информацию. Например, макрокоманды `\hphantom`, `\vphantom` и `\phantom` в приложении B совершенно аналогичны, т.к. автор решил сделать основную часть работы в другой макрокоманде `\phant`, которая является общей для этих трех макрокоманд. Макрокоманде `\phant` каким-то образом должно быть указано, какой вид фантома желателен. Первым приближением было определить команды `\hph` и `\vph` как-нибудь так:

```
\def\hphantom{\ph YN} \def\vphantom{\ph NY} \def\phantom{\ph YY}
\def\ph#1#2{\def\hph{#1}\def\vph{#2}\phant}
```

после чего `\phant` могла проверять `\if Y\hph` и `\if Y\vph`. Это работало, но можно было сделать то же самое более эффективно; например, `\def\hph{#1}` могло быть заменено на `\let\hph=#1`, при этом не будет раскрываться макроопределение. А затем напросилась еще лучшая идея:

```
\def\yes{\if00} \def\no{\if01}
\def\hphantom{\ph\yes\no}...\def\phantom{\ph\yes\yes}
\def\ph#1#2{\let\ifhph=#1\let\ifvph=#2\phant}
```

после чего `\phant` могла проверять `\ifhph` и `\ifvph`. (Эта конструкция была испытана до того, как `\iftrue` и `\iffalse` стали частью языка TEX.) Идея работала

прекрасно, так что автор начал использовать `\yes` и `\no` во многих других ситуациях. Но затем однажды сложное условие обанкротилось, поскольку оно содержало `\ifhph`-подобную проверку внутри других условий:

```
\if...  \ifhph...\fi  ... \else  ...  \fi
```

Вам понятно, какая возникла проблема? Когда выполнялся ⟨истинный текст⟩ самого внешнего условия, все работало прекрасно, поскольку `\ifhph` было либо `\yes`, либо `\no` и это раскрывалось либо в `\if00`, либо в `\if01`. Но когда ⟨истинный текст⟩ был пропущен, `\ifhph` было нераскрыто, поэтому первый `\fi` был ошибочно спарен с первой `\if`, и все перемешалось. Это произошло, когда `\iftrue` и `\iffalse` были вставлены в язык вместо `\yes` и `\no`. Сейчас `\ifhph` является либо `\iftrue`, либо `\iffalse`, поэтому TEX будет правильно его спаривать с закрывающим `\fi` независимо от того, был он пропущен или нет.

Чтобы помочь конструкциям `\if...`, начальный TEX содержит такую макрокоманду `\newif`, что после того, как вы сказали `\newif\ifabc`, будут определены три команды: `\ifabc` (для проверки переключателя), `\abctrue` (для того, чтобы сделать переключатель истинным) и `\abcfalse` (для того, чтобы сделать его ложным). Теперь проблема `\phantom` решается в приложении B так:

```
\newif\ifhph \newif\ifvph
\def\hphantom{\hphtrue\vphfalse\phant}
```

Аналогично определены `\vphantom` и `\phantom`. Это не длиннее того, что было сделано при помощи `\ph`; и снова `\phant` проверяет `\ifhph` и `\ifvph`. Приложение E содержит другие примеры условий, созданных при помощи `\newif`. Новые условия первоначально являются ложными.

Предостережение: не используйте в условном тексте конструкций типа `\let\ifabc=\iftrue`. Если TEX перескочит через эту команду, он подумает, что как `\ifabc`, так и `\iftrue` требуют парных `\fi`, поскольку `\let` не раскрыто! Держите такие команды внутри макроопределений, так чтобы TEX видел `\if...` только тогда, когда он не перескакивает через текст, который читает.

TEX имеет 256 "регистров списков элементов", от `\toks0` до `\toks255`, таких, что списки элементов могут быть перетасованы без передачи через читающий аппарат TEX'а. Есть также такая инструкция `\toksdef`, что, например,

```
\toksdef\catch=22
```

делает `\catch` эквивалентной `\toks22`. Начальный TEX содержит макрокоманду `\newtoks`, которая назначает новый регистр списка элементов и аналогична `\newcount`. Регистры списка элементов работают подобно параметрам списков элементов `\everypar`, `\everyhbox`, `\output`, `\errhelp` и т.д. Для того, чтобы присвоить новое значение параметру или регистру списка элементов, вы задаете либо

⟨переменный элемент⟩`={`⟨текст замены⟩`}`
либо⟨переменный элемент⟩`=`⟨переменный элемент⟩

где ⟨переменный элемент⟩ означает либо параметр списка элементов или команду, определенную при помощи `\toksdef` или `\newtoks`, либо явное обозначение регистра '`\toks`⟨число⟩'.

Те, кто широко использует мощные возможности макроопределений, сталкиваются с ситуациями, когда макроопределения дают неожиданный результат. Мы уже упоминали, что можно задать `\tracingmacros=1` для того, чтобы наблюдать, как TEX раскрывает макрокоманды и какие аргументы он находит. Имеется также другой полезный способ наблюдать за тем, что делает TEX. Если вы установите `\tracingcommands=1`, то, как мы видели в главе 13, TEX будет показывать каждую команду, которую он выполняет. К тому же, если вы установите `\tracingcommands=2`, TEX покажет все условные команды и их результаты, а также безусловные команды, которые в действительности выполняются или раскрываются. Эта диагностическая информация заносится в протокольный файл. Ее также можно увидеть и на терминале, если сказать `\tracingonline=1`. (Кстати, если вы задаете значение `\tracingcommands` больше, чем 2, то получаете ту же информацию, как если бы оно было равно 2.) Аналогично, `\tracingmacros=2` будет отслеживать `\output`, `\everypar` и т. д.

Один из способов понять встречающиеся странности макропроцесса — это использовать только что описанные методы отслеживания и не спеша наблюдать, что и как делает TEX. Другой способ — это выучить правила, по которым раскрываются макрокоманды. Сейчас мы обсудим эти правила.

Жевательный процесс TEX'а превращает входной файл в длинный список элементов, как это объяснялось в главе 8, а его пищеварительные процессы работают только с этим списком элементов. Когда TEX в списке элементов наталкивается на команду, он ищет ее текущее значение и в некоторых случаях, перед тем, как продолжить чтение, раскрывает этот элемент в последовательность других элементов. Процесс раскрытия применяется к макрокомандам и к некоторым специальным примитивам, таким как `\number` и `\if`, которые мы скоро будем рассматривать. Иногда, однако, раскрытие не выполняется; например, когда TEX занимается с `\def`, то ⟨команда⟩, ⟨текст параметров⟩ и ⟨текст замены⟩ этой `\def` не подлежат раскрытию. Аналогично, два элемента после `\ifx` никогда не раскрываются. Полный список тех случаев, когда элементы не раскрываются, будет приведен в этой главе позже. При необходимости можно использовать его для справок.

Теперь давайте рассмотрим команды, которые раскрываются всякий раз, когда раскрытие не подавлено. Такие команды распадаются на несколько классов.

- Макрокоманды. Когда раскрывается макрокоманда, TEX сначала определяет ее аргументы (если они есть), как объяснялось в этой главе ранее. Каждый аргумент является списком элементов; элементы, когда они воспринимаются как аргументы, не раскрываются. Затем TEX замещает макрокоманду и ее аргументы текстом замены.

- Условия. Когда раскрывается `\if...`, TEX читает вперед столько, сколько необходимо, чтобы определить, истинно или ложно условие, и, если оно ложно, то перескакивает вперед (следя за вложением `\if...\fi`) до тех пор, пока не найдет `\else`, `\or` или `\fi`, которые оканчивают пропущенный текст. Аналогично, когда раскрывается `\else`, `\or` или `\fi`, TEX читает до конца текст, который должен был быть пропущен. "Раскрытие" условия является пустым. (Условия всегда уменьшают число элементов, которые видимы на более поздних стадиях

пищеварительного процесса, тогда как макрокоманды обычно увеличивают это число.)

■ `\number`⟨число⟩. Когда TEX раскрывает `\number`, он читает ⟨число⟩, которое за ним следует (последовательно раскрывая элементы). Окончательным раскрытием является десятичное представление этого числа, а если оно отрицательно, то перед ним идет знак "`-`".

■ `\romannumeral`⟨число⟩. Этот случай аналогичен `\number`, но раскрытие состоит из строчных римских цифр. Например, `\romannumeral 1984` производит `mcmlxxxiv`. Раскрытие будет пустым, если число равно нулю или отрицательно.

■ `\string`⟨элемент⟩. TEX сначала читает ⟨элемент⟩ без раскрытия. Если появился элемент команды, его `\string`-раскрытием будет имя команды (включая `\escapechar` как сигнальный символ, если команда — не один активный символ). В противном случае ⟨элемент⟩ — это символьный элемент, а в качестве раскрытого результата сохраняется его символьный код.

■ `\jobname`. Раскрытием является имя, которое TEX выбирает для этой работы. Например, если TEX помещает свой результат в файлы `paper.dvi` и `paper.log`, то `\jobname` раскрывается в `paper`.

■ `\fontname`⟨шрифт⟩. Раскрытием является имя внешнего файла, соответствующее указанному шрифту. Например, `\fontname\tenrm` может раскрыться в `cmr10` (пять элементов). Если шрифт был использован не в своем проектном размере, то в раскрытии появляется и "размер". ⟨Шрифт⟩ — это либо идентификатор, определенный при помощи `\font`, либо `\textfont`⟨число⟩, `\scriptfont`⟨число⟩, или `\scriptscriptfont`⟨число⟩, либо `\font`, который обозначает текущий шрифт.

■ `\meaning`⟨элемент⟩. TEX раскрывает это в ряд символов, которые будут показаны на терминале командами `\let\test=`⟨элемент⟩ `\show\test`. Например, `\meaning A` обычно раскрывается в `the letter A`; `\meaning\A` после `\def\A#1B{\C}` раскрывается в `macro:#1B->\C` .

■ `\csname...\endcsname`. Когда TEX раскрывает `\csname`, он читает парную ему `\endcsname`, по ходу дела раскрывая элементы. После того, как произошло такое раскрытие, должны остаться только элементы символов. Тогда "раскрытием" условного текста `\csname...\endcsname` будет один элемент команды, который, если предварительно не встречался, определен как `\relax`.

■ `\expandafter`⟨элемент⟩. TEX сначала читает элемент, который следует непосредственно после `\expandafter`, не раскрывая его; давайте назовем его элементом t. Затем TEX читает элемент, который идет после t (и, возможно, еще элементы, если этот элемент имеет аргумент), заменяя его на его раскрытие. Наконец TEX ставит t впереди этого раскрытия.

■ `\noexpand`⟨элемент⟩. Раскрытием является сам элемент, но если этот элемент — команда, которая обычно была бы раскрыта по TEXовским правилам раскрытия, он интерпретируется, как если бы его значением было `\relax`.

■ `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark` и `\splitbotmark`.Здесь раскрытием является список элементов в соответствующем "помеченном" регистре (см. главу 23).

■ `\input`(имя файла). Раскрытие нулевое, но TEX приготовился читать из указанного файла перед тем, как рассматривать другие символы из текущего источника.

■ `\endinput`. Раскрытие нулевое. В следующий раз, когда TEX получит конец строки `\input`, он прекратит читать из файла, содержащего эту строку.

■ `\the`⟨внутренняя величина⟩. Раскрытием является список элементов, которые представляют собой текущее значение одной из переменных TEX'а, как это объясняется ниже. Например, `\the\skip5` может быть раскрыто в `5.0pt plus 2.0fil` (17 элементов).

Мощная операция `\the` имеет много подслучаев, которые мы обсудим поочередно. Множество внутренних числовых величин может быть представлено в виде:

■ `\the`⟨параметр⟩, где ⟨параметр⟩ — имя целого параметра TEX'а (например, `\the\widowpenalty`), параметра размеров (например, `\the\parindent`), параметра клея (например, `\the\leftskip`), или параметра математического клея (например, `\the\thinmuskip`).

■ `\the`⟨регистр⟩, где ⟨регистр⟩ — это имя одного из целых регистров TEX'а (например, `\the\count0`), регистров размеров (например, `\the\dimen169`), регистров клея (например, `\the\skip255`) или регистров математического клея (например, `\the\muskip\count2`).

■ `\the`⟨имя кода⟩⟨7-битное число⟩, где ⟨имя кода⟩ — это `\catcode`, `\lccode`, `\uccode`, `\mathcode`, `\sfcode` или `\delcode`. Например, команда `\the\mathcode‘/` дает текущий (целый) математический код для косой черты.

■ `\the`⟨специальный регистр⟩, где ⟨специальный регистр⟩ — это либо целая величина `\prevgraf`, `\deadcycles`, `\insertpenalties` или `\parshape` (число строк в `\parshape`), либо размер `\pagegoal`, `\pagetotal`, `\pagestretch`, `\pagefilstretch`, `\pagefillstretch`, `\pagefilllstretch`, `\pageshrink` или `\pagedepth`. В горизонтальной моде можно получить целое число `\the\spacefactor`, а в вертикальной моде — специальный размер `\the\prevdepth`.

■ `\the\fontdimen`⟨номер параметра⟩⟨шрифт⟩. Это размер. Например, параметром 6 шрифта является значение его “em”, поэтому `\the\fontdimen6\tenrm` дает `10.0pt` (шесть элементов).

■ `\the\hyphenchar`⟨шрифт⟩, `\the\skewchar`⟨шрифт⟩. Это дает соответствующие целые значения, определенные для указанного шрифта.

■ `\the\lastpenalty`, `\the\lastkern`, `\the\lastskip`. Это величина штрафа, керна, клея и математического клея в последнем элементе текущего списка, при условии, что этот элемент является, соответственно, штрафом, керном или клеем; в противном случае результатом будет `0` или `0.0pt`.

■ `\the`⟨определенный символ⟩, где ⟨определенный символ⟩ задан целым числом при помощи `\chardef` или `\mathchardef`. Результатом является это целое значение в десятичной записи.

Во всех случаях, перечисленных до сих пор, `\the` дает результат, который является последовательностью символьных элементов ASCII Каждому элементу присваивается номер категории 12 (“другие”), за тем исключением, что символьный код 32 получает категорию 10 (“пробел”). Это правило используется для присвоения номеров категории элементам, которые получены при помощи `\number`, `\romannumeral`, `\string`, `\meaning`, `\jobname` и `\fontname`.

Есть такие случаи, когда `\the` производит несимвольные элементы, а именно, либо идентификатор шрифта типа `\tenrm`, либо произвольный список элементов:

- `\the`⟨шрифт⟩ производит идентификатор шрифта, который выбирает указанный шрифт. Например, `\the\font`—это команда, соответствующая текущему шрифту.
- `\the`⟨знаковая переменная⟩ дает копию списка элементов, который является текущим значением переменной. Например, можно раскрывать `\the\everypar` и `\the\toks5`.

Примитивная команда TEX'а `\showthe` будет показывать на терминале то, что `\the` производит в раскрытом определении; перед раскрытым результатом идет `>` , а за ним точка. Например, `\showthe\parindent` показывает

```
> 20.0pt.
```

если используется абзацный отступ начального TEX'а.

Приведем обещанный список случаев, когда раскрываемые элементы не раскрываются. Некоторые ситуации содержат примитивы, которые еще не обсуждались, но мы до них в конце концов дойдем. Раскрытие подавляется в следующих случаях:

- Когда элементы удаляются при обнаружении ошибки (см. главу 6).
- Когда элементы пропускаются, поскольку условный текст игнорируется.
- Когда TEX читает аргументы макрокоманд.
- Когда TEX читает команду, определенную при помощи `\let`, `\futurelet`, `\def`, `\gdef`, `\edef`, `\xdef`, `\chardef`, `\mathchardef`, `\countdef`, `\dimendef`, `\skipdef`, `\muskipdef`, `\toksdef`, `\read` или `\font`.
- Когда TEX читает элементы аргументов для `\expandafter`, `\noexpand`, `\string`, `\meaning`, `\let`, `\futurelet`, `\ifx`, `\show`, `\afterassignment` или `\aftergroup`.
- Когда TEX поглощает текст параметров для `\def`, `\gdef`, `\edef` или `\xdef`.
- Когда TEX поглощает текст замены для `\def`, `\gdef` или `\read`, или текст знаковой переменной типа `\everypar` или `\toks0`, или список элементов для `\uppercase`, `\lowercase` или `\write`. (Список элементов для `\write` будет раскрываться позже, когда он выводится в файл.)
- Когда TEX читает преамбулу к выравниванию, кроме как после элемента для примитивной команды `\span`, или когда он читает ⟨клей⟩ после `\tabskip`.
- Сразу после элемента $\$_3$, который начинает математическую моду, чтобы посмотреть, следует ли далее другая $\$_3$.
- Сразу после элемента `_{12}, который начинает алфавитную константу.

Иногда хочется определить новую макрокоманду, текст замены которой раскрывается в зависимости от текущих условий, а не просто дословно копировать текст замены. Для этой цели TEX имеет команду `\edef` (расширенное

определение) и `\xdef` (которая эквивалентна `\global\edef`). Общий формат их такой же, как у `\def` и `\gdef`, но TEX тупо раскрывает элементы текста замены в соответствии с правилами, приведенными выше. Например, рассмотрим

```
\def\double#1{#1#1}
\edef\a{\double{xy}} \edef\a{\double\a}
```

Здесь первый `\edef` эквивалентен `\def\a{xyxy}`, а второй — `\def\a{xyxyxyxy}`. Можно получить все другие виды раскрытия, включая условия; например,

```
\edef\b#1#2{\ifmmode#1\else#2\fi}
```

дает результат, эквивалентный `\def\b#1#2{#1}` если TEX во время `\edef` находится в математической моде; в противном случае результат эквивалентен результату `\def\b#1#2{#2}`.

Расширенные определения, созданные при помощи `\edef` или `\xdef`, продолжают раскрывать элементы до тех пор, пока не останутся только нераскрываемые элементы, за исключением того, что список элементов, полученный при помощи `\the`, далее не раскрывается. Кроме того, элемент, следующий за `\noexpand` тоже не будет раскрываться, поскольку его возможность раскрываться обнулена. Эти две операции могут быть использованы, чтобы управлять тем, что раскрывается, а что нет.

Предположим, например, что вы хотите определить `\a`, которое равно `\b` (раскрываемому), за которым следует `\c` (нераскрываемое), а за ним `\d` (раскрываемое), предполагая, что `\b` и `\d`—простые макрокоманды без параметров. Для этого есть два простых способа:

```
\edef\a{\b\noexpand\c\d}
\toks0={\c} \edef\a{\b\the\toks0 \d}
```

Такой же эффект можно получить, вообще не используя ни `\noexpand`, ни `\the`. Читателю, который хочет больше узнать о механизме раскрывания TEX'а, рекомендуем попробовать выполнить следующие три упражнения.

Упражнение 20.14
Найдите способ определить такое `\a`, как в предыдущем абзаце, не используя примитивы TEX'а `\noexpand` и `\the`.

Упражнение 20.15
Продолжая пример с отменой раскрытия, допустим, что вы хотите раскрыть `\b` полностью так, чтобы остались только нераскрываемые элементы, совсем не хотите раскрывать `\c`, а `\d` хотите раскрыть только на одном уровне. Например, после `\def\b{\c\c}`, `\def\c{*}` и `\def\d{\b\c}` хотелось бы получить такой эффект:`\def\a{**\c\b\c}`. Как, используя `\the`, можно получить такое частичное раскрытие?

Упражнение 20.16
Выполните предыдущее упражнение не используя `\the` или `\noexpand`. (Это непросто.)


Примитивные команды `\mark{...}`, `\message{...}`, `\errmessage{...}`, `\special{...}` и `\write`⟨число⟩`{...}` раскрывают список элементов в скобках почти так же, как `\edef` и `\xdef`. Однако в этих командах символ макропараметра типа `#` не должен повторяться: можно говорить `##` внутри `\edef`, а внутри `\mark` — только `#`. Макрокоманда `\write` — это особый случай, поскольку ее список элементов сначала читается без раскрытия. Раскрытие происходит позже, когда элементы реально записываются в файл.


▸ **Упражнение 20.17**
Сравните следующие два определения:

```
\def\a{\iftrue{\else}\fi}
\edef\b{\iftrue{\else}\fi}
```

Какое из них даст непарную левую фигурную скобку? (Это сложно.)


TEX имеет возможность читать текст сразу из вплоть до 16 файлов вдобавок к тем файлам, которые являются `\input`. Чтобы начать читать такой вспомогательный файл, вы должны сказать

`\openin`⟨число⟩=⟨имя файла⟩

где ⟨число⟩ расположено между 0 и 15. (Начальный TEX связывает числа входных потоков 0 –15 с командой `\newread`, которая аналогична `\newbox`.) В большинстве реализаций TEX'а к имени файла добавляется расширение `.tex`, как и в `\input`, если явно не задано другое расширение. Если файл не найден, TEX не дает сообщения об ошибке, он просто будет считать, что входной поток не открыт. Это условие можно проверить при помощи `\ifeof`. Когда вы закончили работу с файлом, вы можете сказать

`\closein`⟨число⟩

и файл, связанный с этим номером входного потока, закроется, т.е. вернется в свое изначальное состояние, если, конечно, он был открыт. Чтобы вводить из открытого файла, вы говорите

`\read`⟨число⟩`to`⟨команда⟩

и команда становится определенной как макрокоманда без параметров, текст замены которой — это содержание следующей строки, прочитанной из указанного файла. Эта строка преобразуется в список элементов, используя процедуру главы 8, основываясь на текущих номерах категорий. Если необходимо, читаются дополнительные строки до тех пор, пока не встретится равное число левых и правых фигурных скобок. К концу читаемого файла неявно присоединяется пустая строка. Если ⟨число⟩ не расположено между 0 и 15, либо если нет такого открытого файла, либо если файл кончился, ввод будет производиться с терминала: TEX выдаст подсказку для пользователя, если ⟨число⟩ не отрицательно. Макроопределение будет локальным, если не указано `\global\read`.


Например, легко вести диалог с пользователем, используя `\read` вместе с командой `\message`, (которая пишет раскрытый список элементов на

терминал и в протокольный файл):

```
\message{Пожалуйста, напечатайте ваше имя:}
\read16 to\myname
\message{Привет, \myname!}
```

Команда `\read` в этом случае напечатает `\myname=` и будет ждать ответа; ответ будет повторен в протокольном файле; `\myname=` было бы опущено, если вместо `\read16` было бы `\read-1`.

Упражнение 20.18

Только что приведенный пример `\myname` работает не совсем правильно, потому что ⟨return⟩ в конце строки преобразуется в пробел. Как можно устранить этот недостаток?

Упражнение 20.19

Продолжим предыдущий пример. Определите макрокоманду `\MYNAME`, у которой все буквы заглавные. Например, если `\myname` раскрывается в `Артур`, то `\MYNAME` должна раскрываться в `АРТУР`. Предположите, что `\myname` содержит в своем раскрытии только буквы и пробелы.

Приложения B, D и E содержат множество примеров того, как заставить макрокоманды делать всякие полезные вещи. Давайте завершим эту главу, представив несколько примеров того, как можно использовать TEX в качестве примитивного языка программирования, если вам хочется получить специальные эффекты и вы не слишком озабочены стоимостью использования компьютера.

Начальный TEX содержит конструкцию `\loop...\repeat`, которая работает так: вы говорите '`\loop` α `\if...` β `\repeat`', где α и β — любые последовательности команд и где `\if...` — любая проверка условия (без парной `\fi`). TEX сначала выполняет α, затем, если условие истинно, будет выполнять β, а затем повторит весь процесс снова, начиная с α. Если условие окажется ложным, цикл закончится. Например, приведем программу, выполняющую небольшой диалог, в котором TEX ждет, чтобы пользователь отвечал "`Да`" или "`Нет`":

```
\def\yes{Да } \def\no{Нет} \newif\ifgarbage
\loop\message{Вы счастливы? }
  \read-1 to\answer
  \ifx\answer\yes\garbagefalse % ответ:  Да
    \else\ifx\answer\no\garbagefalse % ответ: Нет
      \else\garbagetrue\fi\fi % ответ: чепуха
  \ifgarbage\message{(Пожалуйста, ответьте Да или Нет.)}
\repeat
```

Упражнение 20.20

Используйте механизм `\loop...\repeat` для того, чтобы сделать обобщенную макрокоманду `\punishment`, которая повторяет какой-нибудь абзац любое число раз. Например,

```
\punishment{Я не должен болтать в классе.}{100}
```

должна дать результат, который требовался в упражнении 20.1.


Первыми тридцатью простыми числами являются 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, and 113. Может быть вы не сочтете этот факт поразительным, но вас может удивить, что предыдущее предложение было введено так

```
Первыми тридцатью простыми числами являются \primes{30}.
```

TEX сам сделал все вычисления, раскрывая макрокоманду `\primes`, так что автору приятно быть уверенным, что список простых чисел, приведенных выше, совершенно свободен от опечаток. Приведем набор макрокоманд, которые сделали это:

```
\newif\ifprime \newif\ifunknown % булевские переменные
\newcount\n \newcount\p \newcount\d \newcount\a % целые переменные
\def\primes#1{2,~3% предположим, что #1 не меньше 3
  \n=#1 \advance\n by-2 % n значений для перебора
  \p=5 % нечетные простые числа, начиная с 5
  \loop\ifnum\n>0 \printifprime\advance\p by2 \repeat}
\def\printp{, % мы будем вызывать \printp, если p простое
  \ifnum\n=1 and~\fi % 'и' предшествует последнему значению
  \number\p \advance\n by -1 }
\def\printifprime{\testprimality \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
  \loop\trialdivision \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{\a=\p \divide\a by\d
  \ifnum\a>\d \unknowntrue\else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse\unknownfalse\fi}
```

Вычисления проводятся совершенно прямолинейно, за исключением того, что содержат цикл внутри цикла, поэтому `\testprimality` вводит дополнительный набор фигурных скобок, чтобы внутренний цикл не был перепутан с внешним. Из-за этих скобок необходимо сказать `\global`, когда `\ifprime` устанавливается в истину или ложь. TEX на это предложение потратил больше времени, чем обычно тратит на целую страницу; макрокоманда `\trialdivision` раскрывалась 132 раза.


Макрокоманда `\loop`, которая делает все эти удивительные вещи, в действительности совершенно проста. Она помещает код, который, как предполагается, будет повторяться, в команду, называемую `\body`, а затем повторяется другая команда до тех пор, пока условие не станет ложным.

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body\let\next=\iterate\else\let\next=\relax\fi\next}
```

Раскрытие `\iterate` оканчивается раскрытием `\next`, поэтому TEX может удалить `\iterate` из своей памяти перед тем, как вызывать `\next`, и память в течении длинного цикла не заполняется. Ученые-компьютерщики называют это "хвостовой рекурсией".


Макрокоманда `\hex`, приведенная ниже, которая преобразовывает числовой регистр `\n` в шестнадцатиричную запись, иллюстрирует *рекурсивную* командную структуру, в которой множество копий `\hex` могут действовать одновременно. Для этой цели рекурсия работает лучше, чем простое повторение `\loop`,

поскольку шестнадцатиричные цифры вычисляются справа налево, в то время, как выводиться они должны слева направо. (Число в `\n` должно быть ≥ 0.)

```
\def\hex{{\count0=\n \divide\n by16
  \ifnum\n>0 \hex\fi \count2=\n \multiply\count2 by-16
  \advance\count0 by\count2 \hexdigit}}
\def\hexdigit{\ifnum\count0<10 \number\count0
  \else\advance\count0 by-10 \advance\count0 by`A \char\count0 \fi}
```

Наш последний пример — это макрокоманда, которая вычисляет число непробельных элементов в своем аргументе, например, `\length{аргумент}` раскрывается в 8. Это иллюстрирует еще один аспект применения макроопределений.

```
\def\length#1{{\count0=0 \getlength#1\end \number\count0}}
\def\getlength#1{\ifx#1\end \let\next=\relax
  \else\advance\count0 by1 \let\next=\getlength\fi \next}
```

С этого времени [37 A.D.] влияние Макро стало подавляющим.

By this time [37 A.D.] the influence of Macro had become supreme.
— TACITUS, *Annals* (c. 120 A.D.)

Я ненавижу определения.

I hate definitions.
— BENJAMIN DISRAELI, *Vivian Grey* (1826)

21

Создание боксов

В главах 11 и 12 мы обсуждали концепции боксов и клея и к этому времени уже видели множество приложений этих концепций. В большинстве случаев вы имели дело с боксами, которые автоматически получены TEX'ом при помощи его аппарата для создания абзацев, создания страниц и процессора математических формул, но если вам хочется делать что-либо нестандартное, вы можете создавать боксы и самостоятельно. Например, глава 14 обращает внимание на то, что можно предотвратить перенос слова или расщепление чего-либо между строками, если заключить это в `\hbox`, а глава 19 показывает, что `\hbox` позволяет получать в выделенном уравнении обычный текст.

Цель этой главы — зафиксировать те детали, относящиеся к боксам, которые еще не были описаны. К счастью, осталось обсудить совсем немного. Большинство правил мы уже упоминали, так что эта глава получилась довольно короткой. Действительно, предыдущие главы затрагивали почти все, за исключением правил использования линеек.

Для того, чтобы сделать линейчатый бокс (т. е., сплошной черный прямоугольник), надо ввести либо `\hrule` в вертикальной моде, либо `\vrule` в горизонтальной моде, за которыми следуют либо некоторые, либо все спецификации `width`⟨размер⟩, `height`⟨размер⟩, `depth`⟨размер⟩ в любом порядке. Например, если

```
\vrule height4pt width3pt depth2pt
```

появится в середине абзаца, TEX напечатает черный бокс ▮. Если указать спецификацию дважды, вторая спецификация перекрывает первую. Если размерность остается неуказанной, то по умолчанию получается следующее:

	`\hrule`	`\vrule`
ширина	*	0.4 pt
высота	0.4 pt	*
глубина	0.0 pt	*

Здесь '*' означает, что реальная размерность зависит от контекста: линейка будет расширена до границ наименьшего бокса или выравнивания, в которых она заключена.

Например, автор непосредственно перед этим абзацем ввел `\hrule`, и можно посмотреть, что из этого получилось: горизонтальная линейка толщиной 0.4 pt, растянутая через всю страницу, поскольку вертикальный бокс, который содержит ее, имеет именно такую ширину. (Действительно, содержащий эту линейку вертикальный бокс — это сама страница). Другой пример приведен сразу после этого абзаца. В нем показан результат такой команды:

```
\hrule width5cm height1pt \vskip1pt \hrule width6cm
```

TEX не вставляет междустрочный клей между линейчатыми боксами и их соседями в вертикальном списке, поэтому эти две линейки отстоят друг от друга на 1 pt.

▸ **Упражнение 21.1**
B. L. User не хотел, чтобы одна из его горизонтальных линеек касалась левого поля, поэтому поместил ее в бокс и сдвинул этот бокс вправо так:

```
\moveright 1in \vbox{\hrule width3in}
```

Но оказалось, что пространство над и под линейкой получается больше, чем когда он просто сказал `\hrule width 4in` без всякого `\vbox`. Почему TEX увеличил пробелы, и что надо было бы сделать, чтобы избежать этого?

Если заданы все три размерности линейки, то между командами `\hrule` и `\vrule` нет существенных различий, поскольку обе они создают один и тот же черный бокс. Но если вы хотите вставить этот бокс в вертикальный список, то должны назвать его `\hrule`, а если хотите вставить в горизонтальный список — то `\vrule`, безотносительно к тому, выглядит он реально как горизонтальная или как вертикальная линейка. Если вы в вертикальной моде говорите `\vrule`, TEX начинает новый абзац, а если в горизонтальной моде говорите `\hrule`, TEX прекращает текущий абзац и возвращается в вертикальную моду.

Все размерности линейки могут быть отрицательными. Например, приведем пример линейке, высота которой равна 3 pt, а глубина равна −2 pt: ——. Однако линейка невидима, если ее высота плюс глубина, а также ширина не положительны. Линейку с отрицательной шириной нельзя увидеть, но когда она появляется в горизонтальном списке, она действует как сдвиг влево.

▸ **Упражнение 21.2**
Объясните, как, вероятно, автор в предыдущем абзаце получил линейку ——. [*Намек:* ее длина равна одному дюйму.]

Теперь давайте подытожим все способы явного задания боксов. (1) Символ сам по себе создает символьный бокс в горизонтальной моде; этот символ берется из текущего шрифта. (2) Команды `\hrule` и `\vrule` создают линейчатые боксы, как это только что объяснялось. (3) К тому же можно создавать горизонтальные и вертикальные боксы, для которых используется обобщенный термин ⟨бокс⟩. Бокс может быть одного из следующих семи видов:

`\hbox`⟨спецификация бокса⟩`{`⟨горизонтальный материал⟩`}`	(см. главу 12)
`\vbox`⟨спецификация бокса⟩`{`⟨вертикальный материал⟩`}`	(см. главу 12)
`\vtop`⟨спецификация бокса⟩`{`⟨вертикальный материал⟩`}`	(см. главу 12)
`\box`⟨номер регистра⟩	(см. главу 15)
`\copy`⟨номер регистра⟩	(см. главу 15)
`\vsplit`⟨номер регистра⟩`to`⟨размер⟩	(см. главу 15)
`\lastbox`	(см. главу 21)

Здесь ⟨спецификация бокса⟩ — это либо “`to`⟨размер⟩”, либо “`spread`⟨размер⟩”, либо пусто. Это управляет размещением клея в горизонтальных или вертикальных списках внутри бокса, как объяснялось в главе 12. ⟨Номер регистра⟩ находится между 0 и 255. После того, как вы говорите `\box`, этот регистр становится пустым, но после `\copy` регистр остается неизменным, как объяснялось в главе 15. Операция `\vsplit` также объяснялась в главе 15. В математических модах допустим еще один тип бокса: `\vcenter`⟨спецификация бокса⟩`{`⟨вертикальный материал⟩`}` (см. главу 17).


В приведенной выше таблице нижняя строка содержит `\lastbox`, примитивную операцию, которая прежде не упоминалась. Если последним элементом текущего горизонтального или вертикального списка является h-бокс или v-бокс, он удаляется из списка и превращается в `\lastbox`; в противном случае `\lastbox` пустой. Эта операция разрешена во внутренней вертикальной моде, горизонтальной и ограниченной горизонтальной моде, но ее нельзя использовать для того, чтобы получить бокс из текущей страницы в вертикальной моде. В математических модах `\lastbox` всегда пустой. В начале абзаца `{\setbox0=\lastbox}` убирает бокс абзацного отступа.


Операция `\unskip` похожа на `\lastbox`, за исключением того, что применяется не к боксам, а к клею. Если последний элемент текущего списка — это клей (или проводники, как объясняется ниже), он убирается. Вы не можете убрать клей с текущей страницы в вертикальной моде, сказав `\unskip`, но можете сказать `\vskip-\lastskip`, что обладает почти тем же действием.


В главах с 24 по 26 приведен обзор всех операций TEX'а во всех модах, и когда в этом обзоре упоминается бокс, то подразумевается один из семи только что перечисленных вариантов. Например, в любой моде можно сказать `\setbox`⟨номер регистра⟩`=`⟨бокс⟩, а в вертикальной — `\moveright`⟨размеры⟩⟨бокс⟩. Но нельзя написать `\setbox`⟨номер регистра⟩`=C` или `\moveright`⟨размеры⟩`\hrule`. Если вы попробуете сделать что-нибудь подобное, TEX пожалуется на то, что предполагается, будто бокс присутствует. Символы и линейки настолько специальны, что не считаются боксами.


▸ **Упражнение 21.3**
Определите такую команду `\boxit`, что `\boxit{`⟨бокс⟩`}` дает этот бокс, окруженный трехпунктовыми пробелами и обведенный линиями со всех четырех сторон.

> Например, предложение, которое вы сейчас читаете, напечатано как часть выделенной формулы `$$\boxit{\boxit{\box4}}$$`, где box4 был создан при помощи `\setbox4=\vbox{\hsize 23pc \noindent \strut. Например, предложение, которое вы сейчас читаете ... \strut}`.


Давайте посмотрим, что может находиться внутри бокса. Горизонтальный бокс содержит горизонтальный список, вертикальный — вертикальный список. Оба вида списков составлены в основном из элементов типа боксов, клея, кернов и штрафов, как мы это видели в главах 14 и 15. Но вы можете включать в них и некоторые специальные элементы, которые мы еще не обсуждали, а именно "проводники" ("leaders") и "whatsits." В оставшейся части этой главы мы должны изучить, как использовать такие экзотические элементы.


Точки, которые вы видите, ... называются *проводниками* поскольку они проводят ваши глаза через страницу; такие штуки часто используют в указателях или таблицах содержания. Основная идея — это повторить бокс столько раз, сколько это необходимо, чтобы заполнить некоторое пространство. TEX трактует проводники как специальный случай

клея. Нет, погодите, лучше сказать по-другому: TEX трактует клей как специальный случай проводников. Обычно клей заполняет пространство ничем, в то время как проводники заполняют пространство любым желаемым элементом. В горизонтальной моде можно сказать

```
\leaders⟨бокс или линейка⟩\hskip⟨клей⟩
```

и эффект будет почти тот же, как если бы вы просто сказали `\hskip`⟨клей⟩, за исключением того, что место будет занято копиями указанного бокса или линейки. Клей растягивается или сжимается, как обычно. Например,

```
\def\leaderfill{\leaders\hbox to 1em{\hss.\hss}\hfill}
\line{Альфа\leaderfill Омега}
\line{Введение\leaderfill Заключение}
```

даст следующие две строки:

Альфа . . . . . Омега
Введение . . . . . Заключение

Здесь `\hbox to 1em{\hss.\hss}` задает бокс шириной в один em с точкой в центре. Тогда команда `\leaderfill` указывает, что этот бокс копируется, пока не заполнит пространство в боксе `\line`. (Макрокоманда начального TEX'а `\line` создает h-бокс, ширина которого равна `\hsize`.)

Заметим, что точки в двух строках примера появляются в точности одна под другой. Это не случайность, а следствие того, что операция `\leaders` действует как окно, которое позволяет видеть часть бесконечного ряда боксов. Если слова "Альфа" и "Омега" заменены более длинными словами, число точек может быть другим, но те, которые вы видите, будут на тех же самых местах, что и прежде. Бесконечно повторяемые боксы выстраиваются так, что они касаются друг друга и что если бы вы могли видеть их всех, каждый из них имел бы ту же самую точку привязки, что и наименьший включающий их бокс. Таким образом, `\leaders` поместит бокс вплотную к левому краю охватывающего бокса, если там начинаются проводники, но бокс, придвинутый вплотную вправо не получится, если только ширина охватывающего бокса не делится точно на ширину повторяемого бокса. Если повторяемый бокс имеет ширину w, и если заполняемое пространство как минимум $2w$, то вы всегда будете видеть по меньшей мере одну копию бокса. Но если пространство меньше, чем $2w$, бокс может не появиться, поскольку боксы в бесконечном ряду печатаются только когда их ширина целиком помещается в имеющемся пространстве.

Когда проводники изолированы друг от друга, их не надо выравнивать как это только что описано, поэтому у TEX'а есть также невыровненные проводники. В этом случае бокс шириной w будет скопирован q раз, когда заполняемое пространство не меньше qw и меньше чем $(q+1)w$. Более того, результат будет центрирован в имеющемся пространстве. В TEX'е есть два вида невыровненных проводников, а именно, `\cleaders` (центрированные проводники) и `\xleaders` (расширенные проводники). Центрированные проводники упаковываются в бокс вплотную один к другому, оставляя слева и справа пробелы равной величины; расширенные проводники распределяют дополнительные пробелы поровну между $q+1$ позициями, примыкающими к q боксам. Например, давайте предположим,

что бокс шириной 10 pt используется в проводниках, которыми надо заполнить пространство в 56 pt. Будет использовано пять копий бокса: `\cleaders` произведет пробел шириной 3 pt, затем пять боксов, затем другие 3 pt пробела, а `\xleaders` даст 1 pt пробела, затем первый бокс, затем второй пробел шириной 1 pt, затем второй бокс, ..., затем пятый бокс и 1 pt пробела.

Упражнение 21.4
Предположим, что бокс шириной 10 pt заполняет пространство шириной 38 pt, начинающееся на расстоянии 91 pt от левой границы охватывающего его бокса. Сколько копий этого бокса получится при `\leaders`, `\cleaders`, и `\xleaders`? Как в каждом из этих трех случаев будут расположены боксы относительно левой границы охватывающего их бокса?

Упражнение 21.5
Предполагая, что "." в макрокоманде `\leaderfill` на предыдущей странице имеет ширину только 0.2 em, с обоих краев бокса шириной 1 em располагается пустое место шириной 0.4 em. Поэтому конструкция `\leaders` с каждого края между точками и текстом будет оставлять свободное место величиной от 0.4 em до 1.4 em. Переопределите `\leaderfill` так, чтобы величина пробела с каждого конца была бы между 0.1 em и 1.1 em, но чтобы проводники на соседних строках все еще были выровнены относительно друг друга.

Вместо бокса в конструкции проводников, вы можете задать линейку (либо `\hrule`, либо `\vrule`), со следующими за ней, как обычно, необязательными спецификациями высоты, ширины и глубины. Тогда линейка получится такой же ширины, как соответствующий клей. Это тот случай, когда `\hrule` имеет смысл в горизонтальной моде, поскольку задает горизонтальную линейку в тексте. Например, если в предыдущей иллюстрации макрокоманду `\leaderfill` заменить на

```
\def\leaderfill{ \leaders\hrule\hfill\ }
```

то результат будет выглядеть так:

Альфа _____ Омега
Введение _____ Заключение

Когда вместо бокса используется линейка, она полностью заполняет пространство, поэтому между `\leaders`, `\cleaders` и `\xleaders` нет разницы.

Упражнение 21.6
Что производит `\leaders\vrule\hfill`?

Проводники с тем же успехом работают в вертикальной моде, что и в горизонтальной. В этом случае вместо горизонтального клея используется вертикальный (например, `\vskip`⟨клей⟩ или `\vfill`), а `\leaders` производит боксы, которые выровнены так, что верхушка каждого повторяемого бокса занимает ту же самую вертикальную позицию, что и верхушка наименьшего охватывающего их бокса, плюс кратное величине (высота плюс глубина) повторяемого бокса. В вертикальных проводниках между боксами не вставляется никакого междустрочного клея; боксы надстраиваются прямо один на другой.

Если вы указываете горизонтальные проводники с боксом, ширина которого не положительна или вертикальные проводники с боксом, у которого не положительна высота плюс глубина, TEX молча игнорирует проводники и вместо них производит обычный клей.

Упражнение 21.7
Объясните, как вы можете окончить абзац линейкой, которая имеет длину не меньше 10 pt и простирается вплоть до правых полей: ———————

Горизонтальные проводники несколько отличаются от горизонтального клея тем, что имеют высоту и глубину, которая используется, когда TEX вычисляет размеры охватывающего бокса (даже если число повторений равно нулю). Аналогично, вертикальные проводники имеют ширину.

Упражнение 21.8
Продемонстрируйте, как получить следующий "TEXтуру"

TEXTEXTEXTEXTEXTEXTEXTEXTEXTEX
TEXTEXTEXTEXTEXTEXTEXTEXTEXTEX
TEXTEXTEXTEXTEXTEXTEXTEXTEXTEX
TEXTEXTEXTEXTEXTEXTEXTEXTEXTEX

используя вертикальные проводники внутри горизонтальных. (Эмблема TEX помещается в прямоугольный бокс, и копии этого бокса упаковываются вплотную друг к другу).

Упражнение 21.9
С помощью вертикальных проводников выполните упражнение 20.1.

Макрокоманды `\overbrace` и `\underbrace` начального TEX'а сконструированы комбинацией символов и линеек. Шрифт `cmex10` содержит четыре символа ⏜⏞⏝⏟, каждый из которых имеет нулевую глубину и высоту, равную толщине линейки, которая с ним сочетается. Поэтому легко определить такие макрокоманды `\upbracefill` и `\downbracefill`, что вы, например, можете получить

сказав в вертикальной моде

```
\hbox to 100pt{\downbracefill}\hbox to 50pt{\upbracefill}
```

Подробно эти макроопределения описаны в приложении B.

Определение `\overrightarrow` в приложении B более сложно, чем определение `\overbrace`, поскольку вместо линейки оно включает в себя бокс. Шрифты начального TEX'а сконструированы так, что такие символы, как ← и →, могут быть растянуты при помощи знаков минус; аналогично, ⇐ и ⇒ могут быть растянуты при помощи знаков равенства. Однако вы не можете просто поставить символы один за другим, поскольку между ними остаются зазоры ('←−−' и '⇐=='), поэтому между символами необходимо сделать маленький сдвиг влево. Дополнительные сложности возникают, когда для того, чтобы получить стрелку необходимой длины, требуется повторение нецелого числа знака минус. Для

того, чтобы разрешить эту проблему, макроопределение `\rightarrowfill` в приложении B использует `\cleaders` с повторяемым боксом, содержащим в себе 10 средних кусочков знака минус, где один кусочек равен $\frac{1}{18}$ em. До и после проводников находятся — и →; и достаточно сдвига влево, чтобы компенсировать длину пяти этих кусочков, которую `\cleaders` мог оставить впереди и сзади. В этом случае получается такая макрокоманда, что

```
\hbox to 100pt{\rightarrowfill}
```

дает ————————————→.

Теперь мы о проводниках знаем все. А что мы знаем о whatsits? Whatsits (или "всячина") были созданы как обобщенный механизм, при помощи которого важные специальные приложения трактуются как расширения TEX'a. Системный маг может модифицировать программу TEX, не слишком сильно изменяя коды, так что можно быстро получить новые возможности, вместо того, чтобы кодировать их в макроопределениях. Автор надеется, что такие расширения не будут проводиться слишком часто, поскольку он не хочет, чтобы размножались несовместимые системы псевдо-TEX'a, хотя он понимает, что некоторые специальные книги заслуживают особой обработки. Whatsits дают возможность ввести новые элементы в боксы, не слишком меняя существующие соглашения. Но они делают более трудным перенос программ с одной машины на другую.

Во всех реализациях TEX'a определены два вида whatits. Они в действительности не являются расширениями TEX'a, но кодируются так, как если бы были таковыми, так что их можно считать моделью того, как могли быть сделаны другие расширения. Первый вид связан с выводом текстовых файлов и включает в себя примитивные команды TEX'a `\openout`, `\closeout`, `\write` и `\immediate`. Второй связан со специальными инструкциями, которые могут быть переданы печатающим устройствам через команду TEX'a `\special`.

Возможность писать текстовые файлы, которые могут быть позднее введены другими программами (включая TEX), помогает создавать таблицы содержания, указатели и многое другое. Аналогично командам `\openin` и `\closein` из главы 20 можно сказать `\openout`⟨число⟩=⟨имя файла⟩ и `\closeout`⟨число⟩ , где ⟨число⟩ должно быть от 0 до 15. Если у имени файла нет расширения, оно обычно получает расширение '`.tex`'. Аналогично команде `\read`, которая читает одну строку из файла, существует команда `\write`, которая пишет в файл одну строку. Вы говорите

`\write`⟨число⟩`{`⟨список элементов⟩`}`

и материал отправляется в файл, который соответствует заданному номеру потока. Если ⟨число⟩ отрицательно или больше 15, либо если указанный поток не имеет файла, открытого для вывода, вывод направляется в протокольный файл пользователя и на терминал, если только номер не отрицательный. Начальный TEX содержит команду `\newwrite`, которая назначает выходному потоку номер от 0 до 15. Выходные потоки совершено не зависят от входных потоков.

Однако в действительности вывод происходит в отложенном виде. Команды `\openout`, `\closeout` и `\write`, которые вы даете, не выполняются, когда TEX видит их. Вместо этого TEX помещает эти команды в элементы whatsit

и отправляет их в текущий горизонтальный, вертикальный или математический список, который в это время создается. Никакого реального вывода не произойдет, пока этот whatsit в конце концов не отправится в `dvi`-файл как часть более крупного бокса. Причиной для такой задержки служит то, что `\write` часто используется, чтобы сделать таблицы указателей или содержания и когда инструкция `\write` встречается в середине абзаца, еще не известна точная страница, на которой появится этот конкретный элемент. TEX обычно читает абзац целиком перед тем, как разбивать его на строки и набирает более чем достаточно строк для того, чтобы заполнить страницу, а затем решает, что поместится на странице, как это объяснялось в главах 14 и 15. Поэтому механизм отложенной записи — это надежный способ гарантировать правильность номеров страниц ссылок.

⟨Список элементов⟩ команды `\write` сначала хранится в whatsit без всякого макрорасширения; макрорасширение имеет место позже, когда TEX находится в середине операции `\shipout`. Для примера предположим, что некоторый абзац вашего документа содержит текст

```
...Для \write\inx{пример: \the\count0}примера предположим ...
```

Тогда горизонтальный список для абзаца будет иметь whatsit непосредственно перед словом "`примера`" и сразу после пробела между словами, который следует за "`Для`". Этот элемент whatsit содержит нераскрытый список элементов "`пример: \the\count0`". Он остается в нераскрытом виде в то время, когда абзац разбивается на строки и помещается на текущую страницу. Давайте предположим, что это слово "`пример`" (или некоторая начальная часть его при переносе типа "`при-`") заносится на страницу 256. Тогда TEX запишет в выходной поток `\inx`

```
пример: 256
```

поскольку в это время будет раскрыто `\the\count0`. Конечно, команды `\write` обычно генерируются макроопределениями. Они редко встречаются в тексте в явном виде.

TEX задерживает и команды `\openout` и `\closeout`, также помещая их в whatsits. Таким образом сохраняется относительный порядок команд вывода, если только боксы не вводятся в каком-нибудь другом порядке благодаря вставкам или тому подобным вещам.

Иногда не хочется, чтобы TEX задерживал выполнение команд `\write`, `\openout` или `\closeout`. Можно, например, сказать

```
\shipout\hbox{\write...}
```

но это приведет к нежелательной пустой странице в `dvi`-файле. Поэтому TEX имеет другую возможность, которая обходит эту проблему: если ввести `\immediate` прямо перед `\write`, `\openout` или `\closeout`, то операция будет выполнена немедленно, и не будет создано никакого whatsit. Например,

```
\immediate\write16{До свидания}
```

печатает на терминале "`До свидания`". Без `\immediate`, вы бы не увидели "`До свидания`" до тех пор, пока не выведен текущий список. (А на самом деле вы могли его никогда не увидеть, или увидеть несколько раз, если текущий список

входит в бокс, который копируется.) `\immediate\write16` отличается от `\message`, так как `\write` печатает текст на строке отдельно, а результаты некоторых команд `\message` могли появиться на одной и той же строке, отделенные друг от друга пробелами.


⟨Список элементов⟩ команды `\write` должен быть довольно коротким, поскольку он создает одну строку вывода. Некоторые версии TEX'а не способны писать длинные строки. Если вам надо написать много материала, задайте несколько команд `\write`. Или можно присвоить параметру TEX'а `\newlinechar` номер кода ASCII для некоторого символа, который по вашему желанию будет обозначать "начало новой строки"; тогда TEX каждый раз при выводе этого символа в файл будет начинать новую строку. Например, один из способов вывести две строки на терминал одной командой `\write` — это сказать

```
\newlinechar=`\^^J
\immediate\write16{Две^^Jстроки.}
```

Каждая команда `\write` производит вывод в форме, которую TEX всегда использует, чтобы символически показать список элементов: символы представлены сами собой (за исключением того, что вы получаете сдвоенные символы типа `##` для символов макропараметров); элементы нераскрываемых команд производят их имена, перед которыми стоит `\escapechar`, а за которыми — пробел (если только имя — не активный символ или команда не состоит из одной небуквы).


TEX игнорирует `\write`, `\openout` и `\closeout` whatsits, которые появляются в боксах, направляемых проводниками. Если вы огорчены этим, то напрасно.


Поскольку ⟨список элементов⟩ задержанного `\write` раскрывается в относительно случайный момент времени (когда встречается `\shipout`), надо быть внимательным к тому, какие команды ему разрешается содержать. Часто по отношению к `\write` оказывается кстати метод главы 20 для управления макрорасширениями.


▸ **Упражнение 21.10**
Предположим, вы хотите записать при помощи `\write` список элементов, который включает в себя макрокоманду `\chapno`, содержащую номер текущей главы, а так же `\the\count0`, который ссылается на текущую страницу. Вы хотите, чтобы `\chapno` была раскрыта немедленно, поскольку она может измениться до того, как список элементов будет записан, но чтобы `\the\count0` была раскрыта одновременно с `\shipout`. Как вы можете управлять этим?


Теперь давайте закончим изучение боксов, рассмотрев еще одну возможность. Команда `\special{`⟨список элементов⟩`}` может быть дана в любой моде. Как и `\write`, она помещает свой список в whatsit, и как `\message`, немедленно раскрывает список элементов. Этот список элементов будет выведен в `dvi`-файл вместе с другими печатающими командами, которые организует TEX. Поэтому он неявно связан с определенной позицией на странице, а именно, с точкой привязки, которая была бы представлена, если бы бокс, который появляется вместо whatsit, имел нулевые высоту, глубину и ширину. ⟨Список элементов⟩ в команде `\special`

должен состоять из ключевого слова, за которым, если необходимо, следует пробел или соответствующие аргументы.

```
\special{halftone pic1}
```

может означать, что рисунок файла `pic1` должен быть вставлен на текущей странице, так чтобы его точка привязки находилась в текущей позиции. TEX не рассматривает список элементов, чтобы понять, имеет ли он какой-нибудь смысл; список просто копируется на вывод. Однако надо быть осторожным и не делать список слишком длинным, а то строчечная память TEX'а может быть переполнена. Команда `\special` помогает использовать имеющееся у вас специальное оборудование, например, для печати книг знаменитым TEXниколором.

Программы математического обеспечения, которыми `dvi`-файл преобразовывается в печатаемый результат или на дисплей, должны изящно выходить из положения, когда не могут распознать специальные ключевые слова. Так, операции `\special` никогда не должны делать ничего такого, что изменило бы текущую позицию. Как только вы использовали `\special`, вы получили шанс на то, что выходной файл можно вывести не на всех устройствах вывода, поскольку все функции `\special` являются расширениями TEX'а. Однако автор предвидит, что после тщательных экспериментов в группах пользователей возникнут некоторые стандарты для общих графических операций TEX'а, а затем появится возможность для некоторого единообразия в использовании `\special`-расширений.

Если возраст или немощность исключают кровопролитие, приходится прибегать к заточению.

If age or weaknes doe prohibyte bloudletting, you must use boxing.

— PHILIP BARROUGH, *The Methode of Phisicke* (1583)

Единственное, что никогда не выглядит хорошо — это линейка. Не бывает страниц с линейками, которые без них не выглядели бы лучше.

The only thing that never looks right is a rule. There is not in existence a page with a rule on it that cannot be instantly and obviously improved by taking the rule out.

— GEORGE BERNARD SHAW, in *The Dolphin* (1940)

22

Выравнивание

Наборщики требуют дополнительную плату, когда им приходится печатать таблицы, и для этого есть уважительные причины. Каждая таблица имеет свои особенности, так что над каждой из них необходимо подумать, прокрутить альтернативные подходы, пока не найдется такой вариант, который хорошо смотрится и увязывается с остальным текстом. Однако, не надо бояться делать таблицы при помощи TEX'а, поскольку у начального TEX'а есть "табулирование", которое прекрасно обрабатывает простые ситуации почти так же, как вы выполнили бы их на пишущей машинке. Кроме того, у TEX'а есть мощный механизм выравнивания, который помогает справиться с особо сложной организацией таблиц. Для подавляющего большинства приложений достаточно проанализировать несколько простых случаев таких операций выравнивания.

Давайте сначала рассмотрим табуляцию. Если вы говорите

`\settabs` n `\columns`,

начальный TEX дает возможность получать строки, которые разделены на n колонок равного размера. Каждая строка задается командой

`\+`⟨текст$_1$⟩`&`⟨текст$_2$⟩`&`$\cdots$`\cr`.

При этом ⟨текст$_1$⟩ начинается вплотную к левому полю, ⟨текст$_2$⟩ начинается слева во второй колонке и так далее. Заметим, что строку начинает команда `\+`. За последней колонкой следует команда `\cr`, в которой ветеран узнает аббревиатуру для операции "carriage return" (возврат каретки) на пишущих машинках, имеющих каретку. Например, рассмотрим следующую спецификацию:

```
\settabs 4 \columns
\+&&Текст начинается в третьей колонке\cr
\+&Текст начинается во второй колонке\cr
+\it Текст начинается в первой колонке и&&&
  в четвертой, и&далеко!\cr
```

После инструкции `\settabs4\columns` каждая строка, начинающаяся с `\+`, делится на четыре равные части, так что в результате будет

		Текст начинается в третьей колонке	
	Текст начинается во второй колонке		
Текст начинается в первой колонке и		в четвертой, и	далеко!

Этот пример заслуживает внимательного рассмотрения, поскольку иллюстрирует сразу несколько важных вещей. (1) Знак `&` — это аналог клавиши TAB на многих пишущих машинках. Он указывает TEX'у передвинуться вперед на следующую позицию табуляции. Справа от каждой колонки есть позиция табуляции. В данном примере TEX установил четыре позиции табуляции, обозначенные пунктирными линиями. Пунктирная линия указывает также и границу левого поля, хотя в действительности там

нет табуляции. (2) Но операция `&` не в точности совпадает с операцией TAB механической пишущей машинки, поскольку TEX сначала идет на начало текущей колонки и только потом передвигается на следующую. В таком случае вы всегда можете определить, какую колонку сейчас заполняете, посчитав количество знаков `&`. Это удобно, поскольку иначе при заполнении колонок текстом переменной ширины трудно узнать, прошли ли вы позицию табуляции или нет. Так, на последней строке нашего примера для того, чтобы получить колонку 4, были написаны три знака `&`, хотя текущий текст уже печатался в колонке 2, а возможно, и в колонке 3. (3) Можно сказать `\cr` до того, как вы указали полный набор колонок, если оставшиеся колонки пустые. (4) Операция `&` отличается от TAB еще и в другом отношении. TEX игнорирует пробелы после `&`, следовательно, удобно заканчивать колонку, вводя в конце строки входного файла знак `&` и не беспокоясь, что в этом случае введутся дополнительные пробелы. (Предпоследняя строка примера оканчивается `&`, а за этим символом следует неявный пробел. Если бы TEX не проигнорировал этот пробел, то слова в "в четвертой" не начинались бы в точности в начале четвертой колонки.) Между прочим, начальный TEX игнорирует пробелы и после команд `\+`, так что с первой колонкой обращаются так же, как и с остальными. (5) В последней строке примера команда `\it` указывает, что только первая колонка должна печататься курсивом, несмотря на то, что не было никаких фигурных скобок, чтобы обозначить рамки курсива, поскольку TEX неявно вставляет фигурные скобки вокруг каждого элемента выравнивания.

После заданной однажды команды `\settabs` табуляция остается установленной до тех пор, пока вы не переустановите ее, даже если вы закончили вводить таблицу и как обычно печатаете абзац. Но если вы заключите `\settabs` в фигурные скобки, то табуляция, определенная внутри группы, не действует на табуляцию вне ее; `\global\settabs` не допускается.

Табулированные строки обычно используются между абзацами в тех местах, где бы вы напечатали `\line` или `\centerline`, чтобы получить строки в специальном формате. Но также полезно помещать `\+`-строки в вертикальный бокс: это удобный способ задавать таблицы, которые содержат выровненный материал. Например, если вы вводите

```
$$\vbox{\settabs 3 \columns
  \+Это&странный&пример\cr
  \+выделенного&трехколоночного&формата.\cr}$$
```

то получаете следующее выделение:

Это	странный	пример
выделенного	трехколоночного	формата.

В этом случае первая колонка не прижата вплотную к левым полям, поскольку TEX центрирует выделяемый бокс. Колонки, которые оканчиваются командой `\cr` в строках с `\+`, помещаются в бокс с их естественной шириной, поэтому первая и вторая колонки здесь имеют ширину, равную трети `\hsize`, а третья колонка имеет

ширину, равную ширине слова "пример". В этой конструкции мы использовали `$$`, хотя она не содержит никакой математики, поскольку `$$` служит и другим полезным целям, например, центрирует бокс и вставляет пробелы над и под ним.

Не всегда хочется, чтобы позиции табуляции были на одинаковом расстоянии, поэтому существует другой способ задавать их, вводя

`\+`⟨строка-образец⟩`\cr`

сразу после `\settabs`. В этом случае позиции табуляции помещаются в тех же местах, где знаки `&` в строке-образце, а сама строка-образец не печатается. Например,

```
\settabs\+\indent&Горизонтальные списки\quad&\cr % образец
\+&Горизонтальные списки&Глава 14\cr
\+&Вертикальные списки&Глава 15\cr
\+&Математические списки&Глава 17\cr
```

указывает TEX'у напечатать следующие три строки

Горизонтальные списки	Глава 14
Вертикальные списки	Глава 15
Математические списки	Глава 17

В этом примере команда `\settabs` создает колонку 1, ширина которой равна отступу абзаца; ширина колонки 2 равна ширине текста "Горизонтальные списки" плюс один квадрат. В этом случае установлены только две позиции табуляции, поскольку в строке-образце появляются только два `&`. (Строка-образец тоже могла заканчиваться знаком `&`, поскольку текст, следующий за последним табулятором, не используется.)

Первая строка таблицы не всегда может использоваться в качестве образца, поскольку она не обязательно дает правильные позиции табуляции. Вы должны просмотреть всю таблицу и найти наибольший элемент каждой колонки, а затем составить строку-образец из самого широкого элемента первой колонки, самого широкого элемента второй колонки и так далее, опуская последнюю колонку. Не забудьте в строку-образец включить некоторое дополнительное пространство между колонками, чтобы колонки не касались друг друга.

▸ **Упражнение 22.1**
Объясните, как ввести следующую таблицу [из Beck, Bertholle, and Child, *Mastering the Art of French Cooking* (New York: Knopf, 1961)]:

Вес	*Порции*	*Примерное время приготовления**
8 фунтов	6	1 час 50 – 55 минут
9 фунтов	7 – 8	Около 2 часов
$9\frac{1}{2}$ фунтов	8 – 9	2 часа 10 – 15 минут
$10\frac{1}{2}$ фунтов	9 – 10	2 часа 15 – 20 минут

* Для фаршированного гуся добавьте от 20 до 40 минут.

Если вы хотите поместить элемент вплотную к правой границе колонки, напишите перед ним команду `\hfill`, но не забудьте при этом после него написать `&`, так как TEX будет передвигать информацию до тех пор, пока не коснется следующей позиции табуляции. Аналогично, если вы хотите печатать элемент в центре его колонки, введите `\hfill` перед ним и `\hfill&` после него. Например,

```
\settabs 2 \columns
\+\hfill Этот материал прижат вправо&
    \hfill Этот материал  центрирован\hfill&\cr
\+\hfill в первой части строки.&
    \hfill во второй части строки.\hfill&\cr
```

сделает такую маленькую таблицу:

Этот материал прижат вправо	Этот материал центрирован
в первой части строки.	во второй части строки.

Макроопределение `\+` в приложении B работает так, что помещает ⟨текст⟩ для каждой колонки в горизонтальный бокс следующим образом:

`\hbox to` ⟨ширина колонки⟩`{`⟨текст⟩`\hss}`

Команда `\hss` означает, что текст прижат влево и может продолжаться вправо за границу своего бокса. Поскольку `\hfill` в своей возможности растягиваться "более бесконечна", чем `\hss`, она, как указывалось выше, действует как правое притяжение. Более того, `\hfill` не сжимается, так что TEX будет возражать против переполненного бокса, если что-нибудь не помещается в свою колонку. Можно также центрировать текст, поставив перед ним команду `\hss`, а после него — `&`. В этом случае тексту разрешается высовываться за левую и правую границу своей колонки, и бокс никогда не будет считаться переполненным. Последняя колонка строки `\+` (т. е., элемент, за которым следует `\cr`) будет рассматриваться особо; в него не вставлена команда `\hss` и ⟨текст⟩ просто помещается в горизонтальный бокс со своей естественной шириной.

При печати компьютерных программ встречаются различные трудности, поскольку некоторым авторам нравится использовать стиль, в котором позиции табуляции изменяются от строки к строке. Например, рассмотрим следующий фрагмент программы:

```
if n < r then n := n + 1
         else begin print_totals; n := 0;
              end;
while p > 0 do
  begin q := link(p); free_node(p); p := q;
  end;
```

Здесь устанавливается специальное табулирование, так что **then** и **else** располагаются друг под другом, то же самое касается **begin** и **end**. Это можно сделать, задавая новую строку-образец всякий раз, когда требуется новое расположение табуляторов, но это утомительно, поэтому начальный TEX позволяет более простое решение. Всякий раз, когда вы вводите `&` справа от всех существующих табуляторов, устанавливается новый табулятор таким образом, что только что заполненная

колонка будет иметь свою естественную ширину. Более того, существует операция `\cleartabs` , которая переустанавливает все позиции табуляции вправо от текущей колонки. Поэтому приведенная выше компьютерная программа может быть TEXнически подготовлена следующим образом:

```
$$\vbox{\+\bf if $n<r$ \cleartabs&\bf then $n:=n+1$\cr
  \+&\bf else &{\bf begin} ${\it print\_totals}$; $n:=0$;\cr
  \+&&{\bf end};\cr
  ⟨Оставшаяся часть остается в качестве упражнения⟩}$$
```

▸ **Упражнение 22.2**
Закончите пример компьютерной программы, указав еще три `\+`-строки.

Хотя и разрешено использовать `\+`-строки в вертикальных боксах, нельзя использовать `\+` внутри другой `\+`-строки. Макрокоманда `\+` предназначена только для простых приложений.

Макрокоманды `\+` и `\settabs` приложения B следят за табулированием, поддерживая регистр `\box\tabs` как бокс, заполненный пустыми боксами, ширина которых равна ширине колонок в обратном порядке. Таким образом, можно проверить табуляцию, которая установлена в настоящее время, сказав `\showbox\tabs`. Это записывает ширины колонок в протокольный файл справа налево. Например, после

```
\settabs\+\hskip100pt&\hskip200pt&\cr\showbox\tabs
```

TEX покажет строки

```
\hbox(0.0+0.0)x300.0
.\hbox(0.0+0.0)x200.0
.\hbox(0.0+0.0)x100.0
```

▸ **Упражнение 22.3**
Изучите макрокоманду `\+` в приложении B и придумайте, как можно изменить ее, чтобы табуляция работала, как на механической пишущей машинке (т. е. так, чтобы `&` всегда переводил на следующую позицию табуляции, которая лежит строго справа от текущей позиции). Предполагайте, что пользователь не делает обратный пробел после предыдущих позиций табуляции. Например, если во входном файле у вас `\+&&\hskip-2em&x\cr`, не беспокойтесь, чтобы поместить "x" в первую или вторую колонку, а просто поместите его в начало третьей колонки. (Этот упражнение несколько трудновато.)

У TEX'а есть еще один способ создавать таблицы, используя операцию `\halign` ("горизонтальное выравнивание"). В этом случае формат таблицы основывается не на табулировании, а на записи *шаблона*. Идея заключается в том, чтобы задавать специальное окружение для текста каждой колонки. Конкретные элементы вставляются в свои шаблоны и таблица быстренько заполняется.

Например, давайте еще раз вернемся к рассмотрению примера Горизонтального/Вертикального/Математического списка, который ранее приводился в этой главе. Можно задать его не табулированием, а при помощи команды

`\halign`. Новая спецификация такова

```
\halign{\indent#\hfil&\quad#\hfil\cr
  Горизонтальные списки&Глава 14\cr
  Вертикальные списки&Глава 15\cr
  Математические списки&Глава 17\cr}
```

и это дает тот же самый результат, что и в прошлый раз. Этот пример заслуживает внимательного изучения, поскольку `\halign` окажется совсем простой, как только вы с ней освоитесь. Первая строка содержит *преамбулу* выравнивания, несколько похожую на строку-образец, которая использовалась, чтобы установить позиции табуляции для `\+`. В нашем случае преамбула содержит два шаблона, а именно, `\indent#\hfil` для первой колонки и `\quad#\hfil` для второй. Каждый шаблон содержит в точности один `#`, и это означает: "вставь текст каждого элемента колонки в это место". Таким образом, когда текст "`Горизонтальные списки`" заполняет свой шаблон, первая колонка строки, которая идет за преамбулой, превращается в

```
\indent Горизонтальные списки\hfil
```

а вторая колонка, аналогично, превращается в "`\quad Глава 14\hfil`". Вопрос в том, зачем нужна команда `\hfil`? О, теперь мы дошли до самого интересного места: TEX считывает в свою память целиком всю спецификацию `\halign{...}` до того, как что-либо печатает, и вычисляет максимальную ширину каждой колонки, предполагая, что каждая колонка задана без растяжения или сжатия своего клея. Затем он возвращается назад и помещает каждый элемент в горизонтальный бокс, устанавливая клей так, чтобы каждый отдельный бокс имел максимальную ширину колонки. Вот здесь и появляется `\hfil`: он растягивается, чтобы заполнить дополнительное пространство в соседних элементах.

▸ **Упражнение 22.4**
Какая таблица получится в результате, если в этом примере для первой колонки поставить шаблон "`\indent\hfil#`" вместо "`\indent#\hfil`"?

Перед тем, как читать дальше, проверьте, пожалуйста, правильно ли вы поняли идею шаблонов, описанную в только что приведенном примере. `\halign` и `\+` отличаются в нескольких важных отношениях. (1) `\halign` автоматически вычисляет максимальную ширину колонки; вам не надо искать, какой из элементов будет самым длинным, как в случае, когда вы устанавливали табуляцию при помощи строки-образца. (2) Каждый `\halign` образует колонки своей собственной ширины. Следует специально позаботиться о том, чтобы, если вы этого хотите, две различные операции `\halign` сделали идентичные таблицы. Наоборот, операция `\+` запоминает позиции табуляции до тех пор, пока они специально не переустановлены; между `\+`-строками может появиться любое число абзацев и даже операции `\halign`, не влияя на позиции табуляции. (3) Поскольку `\halign`, чтобы определить максимальную ширину колонки, прочитывает всю таблицу, она не подходит для огромных таблиц, которые занимают несколько страниц книги. Наоборот, операция `\+` имеет дело каждый раз с одной строкой, поэтому не выдвигает никаких специальных запросов на память TEX'а. (Однако, если у вас огромная таблица, вам, возможно, следует определить свою собственную специальную макрокоманду для каждой строки, а не полагаться на общую операцию

`\+`.) (4) `\halign` занимает меньше машинного времени, чем `\+`, поскольку `\halign` — это встроенная команда TEX'а, а `\+` — макрокоманда, которая закодирована в терминах `\halign` и множества других примитивных операций. (5) Шаблоны намного гибче табуляторов, а также помогают экономить время и силы. Например, таблица с Горизонтальным/Вертикальным/Математическим списком может быть задана намного короче, если заметить, что в колонках есть общая информация:

```
\halign{\indent# списки\hfil&\quad Глава #\cr
  Горизонтальные&14\cr Вертикальные&15\cr Математические&17\cr}
```

Можно сэкономить еще два нажатия клавиш, заметив, что все номера глав начинаются с `1`! (Предостережение: дольше придумывать такую оптимизацию, чем печатать таблицу прямолинейным способом; делайте это только тогда, когда вам скучно и хочется чего-нибудь забавного, чтобы поддержать свой интерес.) (6) С другой стороны, шаблоны не заменяют табуляцию, когда позиции табуляции непрерывно меняются, как в примере компьютерной программы.

Давайте сделаем более интересную таблицу, чтобы набраться опыта в работе с `\halign`. Приведем другой пример, из кулинарной книги, цитированной ранее:

Америк. цыплята	*Франц. названия*	*Возраст (месяцы)*	*Вес (фунты)*	*Способы приготовления*
Squab	*Poussin*	2	$^3/_4$ — 1	Гриль, Жаркое
Broiler	*Poulet Nouveau*	2 — 3	$1\,^1/_2$ — $2\,^1/_2$	Гриль, Жаркое
Fryer	*Poulet Reine*	3 — 5	2 — 3	Соте, Жаркое
Roaster	*Poularde*	$5\,^1/_2$ — 9	Свыше 3	Жаркое, Фрикассе
Fowl	*Poule de l'Année*	10 — 12	Свыше 3	Тушить, Фрикассе
Rooster	*Coq*	Свыше 12	Свыше 3	Бульон, Фарш

Заметим, что, за исключением строк заголовка, первая колонка выровнена справа и напечатана жирным шрифтом, средние колонки центрированы, вторая колонка центрирована и напечатана курсивом, а последняя — выровнена слева. Мы бы предпочли вводить ряды таблицы как можно проще, т. е., например, было бы приятно задать нижнюю строку, вводя просто

```
Rooster&Coq&Свыше 12&Свыше 3&Бульон, Фарш\cr
```

и не беспокоясь о стиле печати, выравнивании и так далее. Это не только сокращает количество нажатий клавиш, но также уменьшает вероятность опечатки. Поэтому для первой колонки шаблон должен быть "`\hfil\bf#`", для второй колонки, чтобы получить центрированный текст курсивом, он должен быть "`\hfil\it#\hfil`", и так далее. Нам также нужно разрешить промежутки между колонками, скажем, в один квадрат. *Voila! Вот вам и настольная типография:*

```
\halign{\hfil\bf#&\quad\hfil\it#\hfil&\quad\hfil#\hfil&
        \quad\hfil#\hfil&\quad#\hfil\cr
⟨строчки заголовка⟩
Squab&Poussin&2&\frac3/4 to 1&Гриль, Жаркое\cr
... Фарш\cr}
```

Так же, как в случае операции `\+`, пробелы после `&` игнорируются как в преамбуле, так и в рядах таблицы. Таким образом, когда ряд во входном файле занимает больше одной строки, удобно закончить его знаком `&`.

▸ **Упражнение 22.5**
Как была напечатана строка, содержащая **Fowl**? (Это очень легко.)

В этом примере осталось только задать строки заголовка, формат которых отличается от формата других строк. В данном случае отличается только стиль, поскольку заголовок напечатан наклонным шрифтом, поэтому здесь нет особых трудностей. Вы просто вводите

```
\sl Америк.&\sl Франц.&\sl Возраст&\sl Вес&\sl Способы\cr
\sl цыплята&\sl названия&\sl(месяцы)&\sl(фунты)&\sl приготовления\cr
```

Необходимо каждый раз говорить `\sl`, поскольку каждый элемент таблицы неявно заключен в фигурные скобки.

Автор использовал команду `\openup2pt` для того, чтобы увеличить расстояние между нижними линиями шрифта в этой кулинарной таблице. Внимательный читатель заметит, что между строками заголовка и другими строками есть небольшой дополнительный промежуток. Этот дополнительный промежуток вставлен при помощи `\noalign{\smallskip}` после строки заголовка. Вообще, можно сказать

`\noalign{`⟨материал вертикальной моды⟩`}`

после любого `\cr` в `\halign`; TEX просто скопирует вертикальный материал, не выравнивая его, и этот материал появится, когда `\halign` закончится. Можно использовать `\noalign`, чтобы вставить дополнительный промежуток, как здесь, или вставить штраф, который влияет на разбиение страниц, или даже вставить строку текста (см. главу 19). Определения внутри фигурных скобок в `\noalign{...}` являются локальными для этой группы.

Команда `\halign` также позволяет регулировать промежутки между колонками, чтобы таблица заполняла заданную территорию. Вам не надо решать, что междуколоночный пробел равен одному квадрату, а можно разрешить TEX'у самому принять решение на основании того, какой ширины колонки у него получились, поскольку TEX между колонками помещает "табличный клей". Этот табличный клей обычно равен нулю, но его можно установить в любое значение, которое вам нравится, вводя `\tabskip=`⟨клей⟩. Например, давайте снова создадим кулинарную таблицу, изменив начало спецификации на следующее:

```
\tabskip=1em plus2em minus.5em
\halign to\hsize{\hfil\bf#&\hfil\it#\hfil&\hfil#\hfil&
          \hfil#\hfil&#\hfil\cr
```

Основное тело таблицы не меняется, но из преамбулы убрано `\quad`, а вместо него указан ненулевой `\tabskip`. Кроме того, `\halign` заменен на `\halign to\hsize`. Это означает, что каждая строка таблицы будет помещена в бокс, ширина которого равна текущему значению `\hsize`, т. е., ширине горизонтальной строки, которая

обычно используется в абзаце. Таблица выглядит так:

Америк. цыплята	*Франц. названия*	*Возраст (месяцы)*	*Вес (фунты)*	*Способы приготовления*
Squab	*Poussin*	2	$^3/_4$ — 1	Гриль, Жаркое
Broiler	*Poulet Nouveau*	2 — 3	$1\,^1/_2$ — $2\,^1/_2$	Гриль, Жаркое
Fryer	*Poulet Reine*	3 — 5	2 — 3	Соте, Жаркое
Roaster	*Poularde*	$5\,^1/_2$ — 9	Свыше 3	Жаркое, Фрикассе
Fowl	*Poule de l'Année*	10 — 12	Свыше 3	Тушить, Фрикассе
Rooster	*Coq*	Свыше 12	Свыше 3	Бульон, Фарш

TEX помещает табличный клей перед первой колонкой, после последней колонки и между колонками таблицы. Можно задавать окончательный размер, говоря `\halign to`⟨размер⟩ или `\halign spread`⟨размер⟩ точно также, как указывались `\hbox to`⟨размер⟩ и `\hbox spread`⟨размер⟩. Эта спецификация управляет табличным клеем, но не влияет на распределение клея внутри элементов колонок. (Элементы, как описывалось ранее, уже упакованы в боксы, ширина которых равна максимальной естественной ширине их колонок.)

Поэтому в случае, когда табличный клей не имеет растяжимости или сжимаемости, `\halign to \hsize` ничего не будет делать, кроме того, что укажет TEX'у сообщать о незаполнении или переполнении бокса. Переполнение бокса получается, если табличный клей не может сжаться до величины указанной спецификации. В этом случае вы получаете предупреждение на терминале и в протокольном файле, но в выходном результате переполненный размер таблицы не будет отмечаться "линейкой переполнения". Предупреждающее сообщение показывает "ряд прототипа" (см. главу 27).

Только что приведенный кулинарный пример использовал везде один и тот же табличный клей, но это можно изменить, если изменить `\tabskip` в преамбуле. Перед первой колонкой используется клей, который действует, когда TEX читает "`{`", следующую за `\halign`. Табличный клей, который действует, когда TEX читает `&` после первого шаблона, используется между первой и второй колонками и т.д. Табличный клей, который действует, когда TEX читает `\cr` после последнего шаблона, используется после последней колонки. Например, в

```
\tabskip=3pt
\halign{\hfil#\tabskip=4pt& #\hfil&
  \hbox to 10em{\hss\tabskip=5pt # \hss}\cr ...}
```

преамбула указывает выравнивать строки, которые будут состоять из следующих семи частей:

табличный клей 3 pt;
первая колонка с шаблоном "`\hfil#`";
табличный клей 4 pt;
вторая колонка с шаблоном "`#\hfil`";
табличный клей 4 pt;
третья колонка с шаблоном "`\hbox to 10em{\hss# \hss}`";
табличный клей 5 pt.

TEX копирует шаблоны, не интерпретируя их, за исключением того, что убирает все спецификации клея `\tabskip`. Более точно, знаки преамбулы передаются прямо в шаблоны без макрорасширения; TEX рассматривает только команды `\cr`, `&`, `#`, `\span` и `\tabskip`. Тот ⟨клей⟩, который следует за `\tabskip`, просматривается обычным способом (с макрорасширением), и соответствующие элементы не включаются в текущий шаблон. Заметим, что в приведенном выше примере пробел после `5pt` также исчез. Тот факт, что `\tabskip=5pt` встретился внутри дополнительного уровня фигурных скобок, не сделал определение локальным, поскольку TEX не "видел" этих скобок. Аналогично, если бы перед `\tabskip` находилось `\global`, TEX не сделал бы глобального определения, а только поместил бы `\global` в шаблон. Все присваивания значений `\tabskip` внутри преамбулы являются локальными в `\halign` (за исключением случая, когда `\globaldefs` положительно), поэтому, как только `\halign` завершается, значение `\tabskip` снова будет равно 3 pt.

Когда в преамбуле появляется `\span`, это указывает, что следующий элемент должен быть раскрыт ("ex-span-ded") перед тем, как TEX продолжит чтение.

▸ **Упражнение 22.6**
Сконструируйте преамбулу для следующей таблицы:

Англия	P. Philips	1560–1628	*Нидерланды*	J. P. Sweelinck	1562–1621
	J. Bull	*c*1563–1628		P. Cornet	*c*1570–1633
Германия	H. L. Hassler	1562–1612	*Италия*	G. Frescobaldi	1583–1643
	M. Prætorius	1571–1621	*Испания*	F. Correa de Arauxo	*c*1576–1654
Франция	J. Titelouze	1563–1633	*Португалия*	M. R. Coelho	*c*1555–*c*1635

Табличный клей слева и справа от каждой строки должен быть равен нулю; в центре он должен быть 1 em плюс 2 em; перед именем — .5 em плюс .5 em, а перед датами — 0 em плюс .5 em. Предполагайте, что строки таблицы будут задаваться, например, так,

```
Франция&J. Titelouze&1563--1633&
  Португалия&M. R. Coelho&\\1555--\\1635\cr
```

где `\\` предварительно определено при помощи `\def\\{{\it c\/}}`.

▸ **Упражнение 22.7**
Сконструируйте такую преамбулу, чтобы таблица спряжений

rydw i = I am	**ydw i** = am I	**roeddwn i** = I was
rwyt ti = thou art	**wyt ti** = art thou	**roeddet ti** = thou wast
mae e = he is	**ydy e** = is he	**roedd e** = he was
mae hi = she is	**ydy hi** = is she	**roedd hi** = she was
rydyn ni = we are	**ydyn ni** = are we	**roedden ni** = we were
rydych chi = you are	**ydych chi** = are you	**roeddech chi** = you were
maen nhw = they are	**ydyn nhw** = are they	**roedden nhw** = they were

могла быть задана вводом строк типа

```
mae hi=she is&ydy hi=is she&roedd hi=she was\cr
```

Упражнение 22.8

В таблице, приведенной рядом, TEX выбирает разбиение строк во второй колонке так, что ширина этой колонки равна в точности 16 em. Более того, автор задал один из рядов таблицы, печатая

```
\\393&Plato's {\sl Apology\/};
  Xenophon's
   {\sl Memorabilia\/};
  Aristophanes'
   {\sl Ecclesiazus\ae\/}\cr
```

Можете вы угадать, какая преамбула была использована в выравнивании? [Данные взяты из книги: Will Durant *The Life of Greece* (Simon & Schuster, 1939).]

B.C.
397: War between Syracuse and Carthage
396: Aristippus of Cyrene and Antisthenes of Athens (philosofers)
395: Athens rebuilds the Long Walls
394: Battles of Coronea and Cnidus
393: Plato's *Apology*; Xenophon's *Memorabilia*; Aristophanes' *Ecclesiazusæ*
391–87: Dionysius subjugates south Italy
391: Isocrates opens his school
390: Evagoras Hellenizes Cyprus
387: "King's Peace"; Plato visits Archytas of Taras (mathematician) and Dionysius I
386: Plato founds the Academy
383: Spartans occupy Cadmeia at Thebes
380: Isocrates' *Panegyricus*

Иногда шаблоны будут применяться ко всем элементам колонки, кроме одного или двух. Например, в только что приведенном упражнении двоеточие в первой колонке выравнивания было обеспечено шаблоном "`\hfil#:␣`", но самый первый элемент этой колонки, "B.C.", не должен иметь двоеточия. TEX позволяет избавиться от заданного шаблона следующим способом: если самый первый элемент в элементе выравнивания — это `\omit` (после макрорасширения), то шаблон в преамбуле опускается; вместо него используется простейший шаблон `#`. Например, в приведенную выше таблицу "B.C." было помещено при помощи `\omit\hfil\sevenrm B.C.` сразу после преамбулы. Можно использовать `\omit` в любой колонке, но он в ней должен идти первым, иначе TEX будет вставлять шаблон, который был определен в преамбуле.

Если вы подумаете над тем, что делает TEX, когда выполняет `\halign`, вы поймете, что порядок выполнения некоторых действий имеет принципиальное значение. Макрокоманды не раскрываются, когда читается преамбула, за исключением того, что было описано ранее, но если `\cr` уже однажды встретилось в конце преамбулы, то TEX должен заглянуть вперед, чтобы понять, не является ли следующий элемент `\noalign` или `\omit`, причем макрокоманды раскрываются до тех пор, пока не встретится следующий непробельный элемент. Если этот элемент окажется не `\noalign` или `\omit`, он возвращается назад, чтобы читаться снова, а TEX начинает читать шаблон (к тому же раскрывая макрокоманды). Шаблон состоит из двух частей, части u и части v, где часть u идет перед `#`, а v — после. Когда TEX заканчивает часть u , его читающий механизм возвращается к элементу, который не является ни `\noalign`, ни `\omit`, и продолжает читать элемент таблицы, пока не натолкнется на `&` или `\cr`, которые оканчивают элемент таблицы; затем читается часть v шаблона. В конце части v всегда помещается специальная внутренняя операция, называемая `\endtemplate`. Она указывает TEX'у поместить элемент таблицы в "незаданный бокс", клей которого задастся позже, когда будет известна окончательная ширина колонки. Затем TEX приступает к

чтению другого элемента таблицы: он смотрит, нет ли впереди `\omit` (а также `\noalign` после `\cr`), и процесс продолжается по той же схеме.

Следствием только что описанного процесса является то, что опасно начинать элемент таблицы с `\if...`, или с любой макрокоманды, которая будет раскрываться в текст замены, первым элементом которого будет `\if...`. Причина этого в том, что условие оценивается до того, как будет прочитан шаблон. (TEX еще только смотрит, не встретится ли `\omit`, когда раскрывается `\if`.) Например, если `\strut` определена как аббревиатура для

```
\ifmmode⟨математический текст⟩\else⟨нематематический текст⟩\fi
```

и если `\strut` оказывается первым элементом некоторого элемента выравнивания, то TEX раскроет его в ⟨ нематематический текст⟩, даже если шаблоном будет `$##$`, поскольку TEX не находился в математической моде, когда искал `\omit`. Хаос будет гарантирован. Поэтому в приложении B текстом замены для `\strut` является

```
\relax\ifmmode...
```

и `\relax` также помещается во все другие макроопределения, которые могут пострадать от таких проблем. Иногда нужно, чтобы TEX раскрывал условия перед тем, как вставлен шаблон, но внимательный макроконструктор следит за случаями, в которых это может привести к неприятностям.

Когда вы вводите числовые таблицы, принято располагать десятичные точки в колонке одна под другой. Например, если в одной и той же колонке появляются такие два числа, как 0.2010 и 297.1 надо получить $\begin{matrix}0.2010\\297.1\end{matrix}$. Такой результат не особенно приятен для глаза, но обычно поступают именно так, поэтому, может быть, к этому надо привыкнуть. Это можно сделать так — рассматривать одну колонку как две, так же, как `\eqalign` рассматривает одну формулу как две; точка может быть помещена в начале второй полуколонки. Но автор обычно предпочитает другой, менее искусственный метод, который использует факт, что цифры 0, 1, ..., 9 имеют в большинстве шрифтов одну и ту же ширину: можно выбрать символ, который нигде в таблице не используется, скажем, символ `?`, и изменить его на активный символ, который производит пробел, в точности равный ширине цифры. Затем обычно не трудно поместить такую пустышку в элементы таблицы так, чтобы каждая колонка могла рассматриваться как либо центрированная, либо прижатая вправо или влево. Например, "`??0.2010`" и "`297.1???`" имеют одну и ту же ширину, поэтому их десятичные точки с легкостью выстраиваются одна под другой. Приведем один из способов задания `?` для этой цели:

```
\newdimen\digitwidth
\setbox0=\hbox{\rm0}
\digitwidth=\wd0
\catcode`?=\active
\def?{\kern\digitwidth}
```

Последние два определения будут локальными для некоторой группы,например, внутри `\vbox`, так что "`?`" восстановит свое обычное поведение, когда таблица кончится.

Теперь рассмотрим некоторые математические приложения. Предположим сначала, что вы хотите напечатать маленькую таблицу

$$\begin{aligned} n &= 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ \ 8\ \ 9\ 10\ 11\ 12\ 13\ \ 14\ \ 15\ \ 16\ \ 17\ \ 18\ \ 19\ \ 20\ \ \ldots \\ \mathcal{G}(n) &= 1\ 2\ 4\ 3\ 6\ 7\ 8\ 16\ 18\ 25\ 32\ 11\ 64\ 31\ 128\ 10\ 256\ \ 5\ \ 512\ 28\ 1024\ \ldots \end{aligned}$$

как выделенные формулы. Грубый силовой подход с использованием `\eqalign` или `\atop` обременителен, поскольку $\mathcal{G}(n)$ и n не всегда имеют одно и то же число цифр. Было бы намного приятнее вводить

```
$$\vbox{\halign{⟨преамбула⟩\cr
     n\phantom)&0&1&2&3& ... &20&\dots\cr
     {\cal G}(n)&1&2&4&3& ... &1024&\dots\cr}}$$
```

с некоторой ⟨преамбулой⟩. С другой стороны, ⟨преамбула⟩, несомненно, окажется длинной, так как у этой таблицы 23 колонки, и может показаться, будто `\settabs` и `\+` будет легче. У TEX'а есть удобный механизм, который помогает в таких случаях. Преамбулы часто имеют периодическую структуру, и если вы перед одним из шаблонов поставите дополнительный `&`, TEX будет рассматривать преамбулу как бесконечную последовательность, которая после того, как достигнут `\cr`, начинается снова с помеченного шаблона. Например,

t_1 `&` t_2 `&` t_3 `&&` t_4 `&` t_5 `\cr` трактуется как t_1 `&` t_2 `&` t_3 `&` t_4 `&` t_5 `&` t_4 `&` t_5 `&` t_4 `&` $\cdots$

а

`&`t_1 `&` t_2 `&` t_3 `&` t_4 `&` t_5 `\cr` трактуется как t_1 `&` t_2 `&` t_3 `&` t_4 `&` t_5 `&` t_1 `&` t_2 `&` t_3 `&` $\cdots$.

Табличный клей, следующий за каждым шаблоном, повторяется вместе с этим шаблоном. Преамбула будет расти, насколько это необходимо, в зависимости от количества колонок, которые используются последовательными элементами выравнивания. Поэтому, подходящая ⟨преамбула⟩ для задачи $\mathcal{G}(n)$ — это

```
$\hfil#$ =&\ \hfil#\hfil\cr
```

Теперь предположим, что ваша задача напечатать три пары выделенных формул так, чтобы их знаки = располагались один под другим:

$$\begin{aligned} V_i &= v_i - q_i v_j, \qquad & X_i &= x_i - q_i x_j, \qquad & U_i &= u_i, \qquad \text{для } i \neq j; \\ V_j &= v_j, & X_j &= x_j, & U_j &= u_j + \textstyle\sum_{i\neq j} q_i u_i. \end{aligned} \tag{23}$$

Сделать это при помощи трех `\eqalign` нелегко, поскольку $\sum$ с индексом $i \neq j$ делает правую пару формул больше, чем остальные; базовые линии не будут согласовываны, если только не вставлять "фантомы" в два других `\eqalign` (см. главу 19). Вместо того, чтобы применить `\eqalign`, определенную в приложении B как макрокоманда, которая использует `\halign`, давайте попытаемся применить `\halign` непосредственно. Для этого естественно будет ввести

```
$$\vcenter{\openup1\jot \halign{⟨преамбула⟩\cr
   ⟨первая строка⟩\cr ⟨вторая строка⟩\cr}}\eqno(23)$$
```

поскольку `\vcenter` помещает строки в бокс, который правильно центрирован по отношению к номеру уравнения (23), а макрокоманда `\openup` помещает небольшой дополнительный пробел между строками, как упоминалось в главе 19.


О'кей, а теперь попробуем разобраться, как должна быть сформирована ⟨первая строка⟩ и ⟨вторая строка⟩. Обычно это делается так: перед символами, которые должны располагаться один под другим ставится `&`, так что очевидным решением будет

```
V_i&=v_i-q_iv_j,&X_i&=x_i-q_ix_j,&
  U_i&=u_i,\qquad\hbox{for $i\ne j$};\cr
V_j&=v_j,&X_j&=x_j,&
  U_j&=u_j+\sum_{i\ne j}q_iu_i.\cr
```

Таким образом, выравнивание имеет шесть колонок. Мы можем внести общие символы в преамбулу (например, "`V_`" и "`=v_`"), но это было бы слишком ошибкоопасным и запутанным.


Осталось только сконструировать преамбулу для таких строк. Слева от знаков = нам нужна колонка, дополненная слева, справа от знаков = нам нужна колонка, дополненная справа. Здесь есть небольшая сложность, поскольку, хотя мы и разбили математическую формулу на два отдельных куска, мы хотим иметь такое же распределение пробелов, как если бы это была одна формула. Поскольку мы поставили `&` прямо перед знаком отношения, решением будет вставить `{}` в начале правой формулы. TEX поместит подходящий пробел перед знаком равенства в `${}=...$`, но не поместит его перед знаком равенства в `$=...$`. Поэтому нужная ⟨преамбула⟩ будет такой

```
$\hfil#$&${}#\hfil$&
  \qquad$\hfil#$&${}#\hfil$&
  \qquad$\hfil#$&${}#\hfil$
```

Третья и четвертая колонки такие же, как первая и вторая, за исключением `\qquad`, который разделяет уравнения; пятая и шестая колонки такие же, как третья и четвертая. Мы можем снова использовать удобное сокращение `&&`, чтобы уменьшить преамбулу до

```
$\hfil#$&&${}#\hfil$&\qquad$\hfil#$
```

После небольшой практики вы обнаружите, что вам стало легко придумывать нужные вам преамбулы. Однако, в большинстве рукописей преамбулы не нужны, так что может быть вы очень долго не сможете получить даже этой небольшой практики.


▸ **Упражнение 22.9**
Объясните, как получить следующие выделенные формулы:

$$10w + \ 3x + 3y + 18z = 1, \tag{9}$$
$$6w - 17x \qquad - \ 5z = 2. \tag{10}$$


Следующий уровень сложности возникает, когда некоторые элементы таблицы занимают две или более колонки. Для таких случаев у TEX'а есть два способа. Первый — это `\hidewidth`, который определяется начальным TEX'ом как эквивалент

```
\hskip-1000pt plus 1fill
```

Другими словами, `\hidewidth` имеет экстремально отрицательную "естественную ширину", но может растягиваться без ограничения. Если вы вставите `\hidewidth` справа от некоторого элемента в выравнивании, то получаете такой эффект: игнорируется ширина этого элемента и ему позволяется высовываться вправо из своего бокса. (Подумайте об этом; этот элемент уже не будет самым широким, когда `\halign` будет вычислять ширину колонки.) Аналогично, если вы поставите `\hidewidth` слева от элемента, он сможет высовываться влево, и, как вы увидите позже, можно поставить `\hidewidth` и слева, и справа.

Второй способ управлять элементами таблицы, занимающими более одной колонки — это использовать примитив `\span`, который может быть использован вместо `&` в любой строке таблицы. (Мы уже видели, что `\span` в преамбуле означает "expand", т.е. "расширить"), но вне преамбулы он используется совершенно иначе. Когда вместо `&` появляется `\span`, то материал до и после `\span` обрабатывается обычным образом, но затем помещается не в два бокса, а в один. Шириной этого составного бокса является сумма ширины отдельных колонок плюс ширина табличного клея между ними, поэтому объединенный бокс будет вертикально выровнен с необъединенными боксами других рядов.

Например, предположим, что есть три колонки с, соответственно, шаблонами u_1 `#` v_1 `&` u_2 `#` v_2 `&` u_3 `#` v_3. Предположим также, что ширины колонок равны w_1, w_2, w_3 и что g_0, g_1, g_2, g_3 — это ширины табличного клея после того, как клей установлен. Также предположим, что в выравнивании появилась строка

a_1`\span` a_2`\span` a_3`\cr`

Тогда материал "$u_1a_1v_1u_2a_2v_2u_3a_3v_3$" (т. е., результат "$u_1a_1v_1$" колонки 1 и результаты колонок 2 и 3) будет помещен в h-бокс шириной $w_1 + g_1 + w_2 + g_2 + w_3$. Перед этим боксом в более крупном боксе, который содержит целую выровненную строку, будет расположен клей шириной g_0, а после него — клей шириной g_3.

Можно использовать `\omit` вместе с `\span`. Например, если продолжить пример предыдущего абзаца, то строка

`\omit` a_1 `\span` a_2 `\span\omit` a_3 `\cr`

поместит материал "$a_1u_2a_2v_2a_3$" в h-бокс, который только что рассматривался.

Часто приходится сливать несколько колонок и опускать все их шаблоны, поэтому начальный TEX имеет макрокоманду `\multispan`, которая соединяет заданное число колонок. Например, команда `\multispan3` раскрывается в `\omit\span\omit\span\omit`. Если число соединяемых колонок больше 9, мы должны заключить его в фигурные скобки, например, `\multispan{13}`.

Предыдущие абзацы несколько абстрактны, поэтому давайте рассмотрим пример, который показывает, как реально работает `\span`. Предположим, вы вводите

```
$$\tabskip=3em
\vbox{\halign{&\hrulefill#\hrulefill\cr
    первая&вторая&третья\cr
    первая-и-вторая\span\omit&\cr
    &вторая-и-третья\span\omit\cr
    первая-вторая-третья\span\omit\span\omit\cr}}$$
```

Преамбула задает произвольное число шаблонов `\hrulefill#\hrulefill`; макрокоманда `\hrulefill` аналогична `\hfill`, только пространство заполняется горизонтальной линейкой. Поэтому в результирующей таблице можно увидеть такое заполнение элементов, которое показывает соединенные колонки:

первая	вторая	третья
первая-и-вторая		
	вторая-и-третья	
первая-вторая-третья		

Линейки оканчиваются там, где колонки разделяет табличный клей. Линеек нет в первой строке, поскольку элементы этой строки были в своих колонках самыми широкими. Однако, если бы междустрочный клей был равен 1 em , а не 3 em, таблица выглядела бы так:

первая	вторая	третья
первая-и-вторая		
	вторая-и-третья	
первая-вторая-третья		

Упражнение 22.10

Рассмотрите следующую таблицу, которая называется ведомостью Уолтера:

1 Установленный общий доход		$4,000
2 Нульскобочная сумма для данного лица	$2,300	
3 Заработанный доход ...	1,500	
4 Разность строки 3 и строки 2.....		800
5 Сложи строки 1 и 4. Впиши здесь и на бланке 1040, строка 35		$4,800

Определите преамбулу, чтобы эта ведомость вводилась так:

```
\halign{⟨преамбула⟩\cr
  1&Установленный общий доход\dotfill\span\omit\span&\$4,000\cr
  2&Нульскобочная сумма&\cr
   &для данного лица\dotfill\span\omit&\$2,300\cr
  3&Заработанный доход\dotfill\span\omit&\underbar{ 1,500}\cr
  4&Разность строки 3 и строки 2\dotfill
       \span\omit\span&\underbar{ 800}\cr
  5&Сложи строки 1 и 4. Впиши здесь\span\omit\span\cr
   &и на бланке 1040, строка 35\dotfill\span\omit\span&\$4,800\cr}
```

(Макрокоманда `\dotfill` аналогична `\hrulefill`, но заполнение происходит точками; макрокоманда `\underbar` помещает свой аргумент в h-бокс и подчеркивает его)

Заметим "раннее" появление `\cr` в строке 2 предыдущего примера. Иногда не надо в каждой строке таблица иметь одно и то же количество колонок; `\cr` означает, что в текущей строке колонок больше нет.

▸ **Упражнение 22.11**
Объясните, как ввести обобщенную матрицу $\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots\ldots & \ldots\ldots & \ldots & \ldots\ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$.

Присутствие объединенных колонок усложняет правила TEX'а, по которым он вычисляет ширину колонок. Вместо того, чтобы просто выбрать максимальную естественную ширину элементов колонки, надо быть уверенным, что сумма некоторых ширин достаточно велика, чтобы вместить объединенные элементы. Поэтому опишем здесь то, что TEX реально делает. Во-первых, если некоторая пара всегда объединяется вместе (т. е., если между ними всегда `\span`, когда используется хотя бы один из них), то эти две колонки в действительности сливаются в одну и клей между ними устанавливается равным нулю. Это сводит задачу к случаю, когда каждая позиция табуляции приходится на границу. Пусть после такого сведения осталось n колонок и пусть для $1 \le i \le j \le n$ w_{ij} — максимальная естественная ширина всех элементов, которые соединяются из колонок от i до j, включительно. Если нет таких соединенных элементов, пусть $w_{ij} = -\infty$. (Объединение зависимых колонок гарантирует, что для каждого j существует такое $i \le j$, что $w_{ij} > -\infty$.) Пусть t_k — естественная ширина табличного клея между колонками k и $k+1$ для $1 \le k < n$. Тогда окончательная ширина w_j колонки j определяется по формуле

$$w_j = \max_{1\le i\le j}\bigl(w_{ij} - \textstyle\sum_{i\le k<j}(w_k + t_k)\bigr)$$

для $j = 1, 2, \ldots, n$ (в этом порядке). Отсюда следует, что $w_{ij} \le w_i + t_i + \cdots + t_{j-1} + w_j$ для всех $i \le j$, что и требуется. После того, как определены ширины w_j, табличный клей может быть должен растягиваться или сжиматься. Если он сжимается, значит w_{ij} оказался больше, чем окончательная ширина бокса, который объединяет колонки от i до j, следовательно клей в таком боксе должен был сжаться.

Обычно эти формулы работают прекрасно, но иногда дают нежелательный эффект. Например, предположим, что $n = 3$, $w_{11} = w_{22} = w_{33} = 10$, $w_{12} = w_{23} = -\infty$, и $w_{13} = 100$; другими словами, колонки сами по себе довольно узкие, но существует элемент большой ширины, который предполагает соединить все три колонки. В этом случае формулы TEX'а дают $w_1 = w_2 = 10$, а $w_3 = 80 - t_1 - t_2$, поэтому вся лишняя ширина присоединяется к третьей колонке. Если это вас не устраивает, надо использовать `\hidewidth`, или увеличить естественную ширину табличного клея между колонками.

Следующий уровень сложности, который встречается в таблицах — это горизонтальные и вертикальные линии. Тех, кто умеет разлиновывать таблицы, можно считать Мастерами TEX'а. Приготовились?

При неправильном подходе к вертикальным линиям они могут показаться трудными. Но *можно* получить их, не проливая слишком много слез. Первым делом надо сказать `\offinterlineskip`, что означает, что между строками не будет пустого места. TEX не может позволить вставлять междустрочный клей обычным умным способом, поскольку предполагается, что каждая строчка содержит `\vrule`, которая примыкает к другим `\vrule` соседних строк выше и/или ниже этой. Мы будем в каждую строку вставлять подпорку (strut), включая ее

в преамбулу; тогда у каждой строки будет подходящая высота и глубина, и междустрочный клей будет не нужен. TEX помещает каждый элемент выравнивания в h-бокс, высота и глубина которого устанавливаются равными высоте и глубине всей строки; поэтому команды `\vrule` будут растягивать строки вверх и вниз, даже когда их высота и/или глубина не заданы.

Каждой вертикальной линии должна быть назначена "колонка", и такой колонке может быть приписан шаблон `\vrule#`. Тогда вы получаете вертикальную линейку, просто оставляя элементы этой колонки пустыми в обычных строках выравнивания, или, если вы хотите опустить линейку в какой-нибудь строке, вы должны сказать `\omit`, или можете сказать `height 10pt`, если хотите иметь нестандартную высоту и т.д.

Приведем маленькую таблицу, которая иллюстрирует то, что только что описано. [Данные взяты из статьи A. H. Westing, *BioScience* **31** (1981), 523–524.]

```
\vbox{\offinterlineskip
\hrule
\halign{&\vrule#&
  \strut\quad\hfil#\quad\cr
height2pt&\omit&&\omit&\cr
&Год\hfil&&Население Земли&\cr
height2pt&\omit&&\omit&\cr
\noalign{\hrule}
height2pt&\omit&&\omit&\cr
&8000\BC&&5,000,000&\cr
&50\AD&&200,000,000&\cr
&1650\AD&&500,000,000&\cr
&1850\AD&&1,000,000,000&\cr
&1945\AD&&2,300,000,000&\cr
&1980\AD&&4,400,000,000&\cr
height2pt&\omit&&\omit&\cr}
\hrule}
```

Год	Население Земли
8000 B.C.	5,000,000
50 A.D.	200,000,000
1650 A.D.	500,000,000
1850 A.D.	1,000,000,000
1945 A.D.	2,300,000,000
1980 A.D.	4,400,000,000

В этом примере первая, третья и пятая колонки зарезервированы для вертикальных линий. Горизонтальные линии получаются, если сказать `\hrule` вне `\halign` или `\noalign{\hrule}` внутри его, поскольку `\halign` появляется в v-боксе, ширина которого равна полной ширине таблицы. Горизонтальные линии также можно было бы задать, если сказать `\multispan5\hrulefill` внутри `\halign`, поскольку это дает линию, которая соединяет все пять колонок.

В этой таблице осталась только одна неочевидная вещь — это содержащиеся в ней несколько строчек `height2pt&\omit&&\omit&\cr`. Вам понятно, что они делают? Инструкции `\omit` означают, что нет числовой информации, а также подавляют `\strut` в строках; `height2pt` создает первую `\vrule` высотой 2 pt, и две другие линии будут такими же. Таким образом, действием этих строк будет продолжение вертикальных линий на два пункта, после чего они коснутся горизонтальных линий, этот легкий штрих, которое улучшает внешний вид таблицы. Посмотрите на нее как на образец отличного качества.

Упражнение 22.12
Объясните, почему при вводе этой таблицы сказано `&\cr`, а не просто `\cr`.

Другой способ получить в таблицах вертикальные линии — это ввести таблицу без них, а затем вернуться назад, используя отрицательный клей, и вставить линии.

Приведем другую таблицу. Она стала классической с тех пор, как Michael Lesk опубликовал ее как один из первых примеров в своей статье о форматировании таблиц [Bell Laboratories Computing Science Technical Report **49** (1976)]. Эта таблица иллюстрирует некоторые типичные проблемы, которые возникают в связи с боксированной информацией. Чтобы продемонстрировать способность TEX'а адаптировать таблицу к различным обстоятельствам, табличный клей используется здесь для регулирования ширины колонок. Таблица приведена дважды; один раз она генерирована с `\halign to125pt`, а другой — с `\halign to200pt`, ничего больше не меняя.

Акции AT&T		
Год	Цена	Дивиден.
1971	41–54	$2.60
2	41–54	2.70
3	46–55	2.87
4	40–53	3.24
5	45–52	3.40
6	51–59	.95*

* (в первом квартале)

Акции AT&T		
Год	Цена	Дивиден.
1971	41–54	$2.60
2	41–54	2.70
3	46–55	2.87
4	40–53	3.24
5	45–52	3.40
6	51–59	.95*

* (в первом квартале)

Это сделал следующий набор команд:

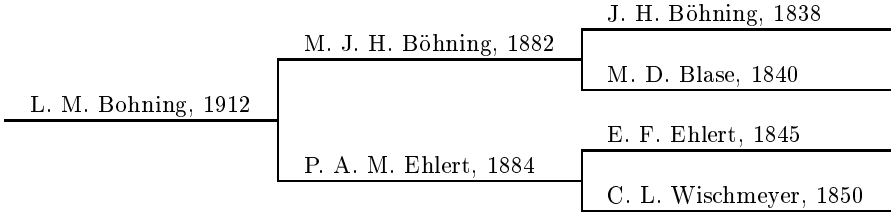
```
\vbox{\tabskip=0pt \offinterlineskip
\def\tablerule{\noalign{\hrule}}
\halign to⟨dimen⟩{\strut#& \vrule#\tabskip=1em plus2em&
  \hfil#& \vrule#& \hfil#\hfil& \vrule#&
  \hfil#& \vrule#\tabskip=0pt\cr\tablerule
&&\multispan5\hfil Акции AT\&T\hfil&\cr\tablerule
&&\omit\hidewidth Год\hidewidth&&
 \omit\hidewidth Цена\hidewidth&&
 \omit\hidewidth Дивиденты\hidewidth&\cr\tablerule
&&1971&&41--54&&\$2.60&\cr\tablerule
&&   2&&41--54&&2.70&\cr\tablerule
&&   3&&46--55&&2.87&\cr\tablerule
&&   4&&40--53&&3.24&\cr\tablerule
&&   5&&45--52&&3.40&\cr\tablerule
&&   6&&51--59&&.95\rlap*&\cr\tablerule \noalign{\smallskip}
&\multispan7* (только в первом квартале)\hfil\cr}}
```

Здесь интересно вот что. (1) Первая колонка содержит подпорку, иначе было бы необходимо ставить подпорки на строках, которые говорят "AT&T" и "(только в первом квартале)", поскольку эти строки опускают шаблоны других колонок,

которые могли содержать встроенные шаблоны. (2) В строке заголовка использовано `\hidewidth`, так что на ширину колонок будет влиять только ширина числовых данных. (3) Использована команда `\rlap`, так что звездочки не влияют на выравнивание чисел. (4) Если бы в спецификации табличного клея было бы `0em plus3em` вместо `1em plus2em`, выравнивание не получилось бы правильным, поскольку "Акции AT&T" были бы шире, чем естественная ширина всего, что они объединяют; вся дополнительная ширина пришлась бы на колонку "Дивиденты".

Упражнение 22.13
Объясните, как добавить по 2 pt дополнительного пространства выше и ниже "Акции AT&T".

Упражнение 22.14
Напечатайте следующую схему, делая ее ширину в точности 35em:



Если у вас неприятности с отладкой таблицы, иногда помогает поставить `\ddt` в начало и конец шаблонов в преамбуле. Это неопределенная команда, которая указывает TEX'у остановиться и показать остаток шаблона. Когда TEX останавливается, можно использовать команды `\showlists` и другие, чтобы увидеть, что машина собирается делать. Если же TEX не остановится, вы узнаете, что не дошли до этой части шаблона.

Можно иметь выравнивание внутри выравнивания. Поэтому когда TEX видит `&`, `span` или `\cr`, ему нужен какой-нибудь способ, чтобы решить, к какому выравниванию это относится. Правило такое: элемент выравнивания оканчивается тогда, когда `&`, `\span` или `\cr` встретятся на том уровне скобок, который был текущим, когда элемент начинался, т.е., в каждом элементе выравнивания должно быть равное число правых и левых фигурных скобок. Например, в строке

```
\matrix{1&1\cr 0&1\cr}&\matrix{0&1\cr 0&0\cr}\cr
```

TEX не возобновляет шаблон для правой колонки, когда сканирует аргумент для `\matrix`, поскольку знаки `&` и `\cr` в этом аргументе заключены в фигурные скобки. Аналогично, знаки `&` и `\cr` в преамбуле не обозначают конец шаблона, если результирующий шаблон не имеет равное количество левых и правых фигурных скобок.

Надо быть внимательным при использовании `&`, `\span` и `\cr`, поскольку эти знаки воспринимаются сканером TEX'а, даже когда он не раскрывает макрокоманду. Например, если вы скажете `\let\x=\span` в середине элемента выравнивания, TEX подумает, что `\span` заканчивает элемент, поэтому `\x` будет равным первому элементу, который в шаблоне следует за `#`. Можно спрятать эту команду `\span` в фигурные скобки, например, `{\global\let\x=\span}` (а приложение D объясняет, как здесь обойтись без `\global`.)


Иногда забывают ставить `\cr` в последней строке выравнивания. Это может привести к фантастическим эффектам, поскольку TEX не ясновидец. Например, рассмотрим следующий простейший случай:

```
\halign{\centerline{#}\cr
  Центрированная строка.\cr
  А другая?}
```

(Заметьте пропущенный `\cr`.) Здесь, когда TEX читает ошибочную строку, происходит забавная штука, поэтому, пожалуйста, будьте внимательны. Шаблон начинается с `\centerline{` , поэтому TEX начинает просматривать аргумент для `\centerline`. Поскольку здесь нет `\cr` после знака вопроса, то `}` после знака вопроса рассматривается как конец аргумента `\centerline`, а *не* как конец аргумента `\halign`. TEX не может закончить выравнивание, если последующий текст не имеет вид "`...{...\cr`". Конечно, элемент типа `a}b{c` является законченным по отношению к шаблону `\centerline{#}`, поскольку это производит `\centerline{a}b{c}`. TEX прав, когда в этом случае не дает сообщения об ошибке. Но решение компьютера в такой ситуации отличается от задумки пользователя, так что запутывающее сообщение об ошибке, вероятно, встретится несколькими строками позже.


Для того, чтобы помочь справиться с такой ситуацией, существует примитивная команда `\crcr`, которая действует также, как `\cr`, но не делает ничего, когда за ней следует `\cr` или `\noalign{...}`. Таким образом, когда вы пишете макрокоманду типа `\matrix`, вы можете спокойно вставить `\crcr` в конце аргумента: это прикроет ошибку, если пользователь забыл конечный `\cr`, и не принесет никакого вреда, если конечный `\cr` присутствует.


Вы устали вводить `\cr`? Можно сделать так, чтобы начальный TEX вставлял `\cr` автоматически в конце каждой входной строчки:

```
\begingroup \let\par=\cr \obeylines %
\halign{⟨преамбула⟩
  ⟨первая строка выравнивания⟩
  ...
  ⟨последняя строка выравнивания⟩
  }\endgroup
```

Это работает, потому что `\obeylines` превращает символ ASCII ⟨return⟩ в активный символ, который использует текущее значение `\par`, и начальный TEX помещает ⟨return⟩ в конце входной строки (см. главу 9). Если вы не хотите иметь `\cr` в конце некоторой строки, введите `%`, и соответствующее `\cr` будет "закомментировано". (Эта специальная мода не работает с `\+`-строками, поскольку `\+` является макрокомандой, аргумент которой ограничен элементом `\cr`, а не просто элементом, который имеет то же значение, что и `\cr`. Но если хотите, вы можете переопределить `\+`, чтобы преодолеть этот барьер. Например, определить макрокоманду `\alternateplus`, которая почти такая же, как `\+`, за исключением того, что ее аргумент ограничивается активным символом `^^M`, а затем включить команду `\let\+=\alternateplus` как часть `\obeylines`.)


Команда `\valign` аналогична `\halign`, но ряды и колонки меняются ролями. В этом случае `\cr` отмечает низ колонки, а выровненные колонки

являются вертикальными боксами, которые собираются вместе в горизонтальной моде. Конкретные элементы каждой колонки помещены в v-боксы с нулевой глубиной. (т. е., как если бы `\boxmaxdepth` была нулевой, как объяснялось в главе 12); высоты элементов для каждого ряда максимизируются таким же образом, как максимизировались ширины элементов для каждой колонки в `\valign`. Операция `\noalign` теперь может быть использована, чтобы вставить материал горизонтальной моды между колонками; операция `\span` теперь соединяет ряды. Обычно приходится работать с TEX'ом по меньшей мере год, прежде чем придется впервые применить `\valign`, и тогда это обычно однорядный `\valign{\vfil#\vfil\cr...}`. Но здесь приведен общий механизм на случай, если он понадобится.

Шестнадцать пенни расставляются
в виде квадрата,
с одинаковым числом пенни
в каждом ряде, в каждой колонке,
и на каждой диагонали.
Можно сделать то же самое
с двенадцатью пенни?

If sixteen pennies are arranged
in the form of a square
there will be the same number
of pennies in every row, every column,
and each of the two long diagonals.
Can you do the same
with twenty pennies?

—HENRY ERNEST DUDENEY, *The Best Coin Problems*(1909)

Именно она управляла
всей Пятой Колонной.

It was she who controlled
the whole of the Fifth Column.

— AGATHA CHRISTIE, *N or M?* (1941)

23

Программы вывода

В главе 15 мы изучали, как TEX строит таблицы, и обсуждали основную двухступенчатую стратегию, которая им используется: TEX собирает материал, пока его не накопится больше, чем может поместиться на странице, затем он выбрасывает одну страницу текста, основанную на том, что, как ему кажется, является самой лучшей точкой разбиения между страницами; затем возвращается и таким же способом собирает материал для следующей страницы. Номера страниц, заголовки и тому подобные вещи присоединяются после того, как каждая страница сброшена, при помощи специальной последовательности команд TEX, которая называется текущей *программой вывода*.

Начальный TEX имеет программу вывода, которая выполняет обыденную работу. Она делает простые операции, которые требуются в большинстве рукописей, а также справляется и с более сложными, такими, как создание вставок при помощи `\footnote` и `\topinsert`, как это было описано в разделах главы 15, помеченных знаками опасного поворота. Мы начнем эту главу с обсуждения, как сделать простые изменения в поведении программы вывода начального TEX'а, а затем разберемся, как определить программу вывода, чтобы она решала более сложные задачи.

Если вы запускаете TEX без изменения формата начального TEX'а, то получаете страницы с номерами внизу. Каждая страница будет иметь ширину приблизительно $8\frac{1}{2}$ дюйма и высоту приблизительно 11 дюймов, включая поля в 1 дюйм со всех четырех сторон. Этот формат подходит для препринтов технических статей, но если вы используете TEX для других целей, вам очень даже может захотеться изменить его.

Например, мы видели в экспериментах главы 6, что можно изменить ширину страницы, если задавать различное значение размера горизонтальной строки `\hsize`. Формат начального TEX говорит `\hsize=6.5in`, чтобы получить страницу в 8.5 дюйма с 1-дюймовыми полями, но вы, если хотите, можете изменить это значение. Аналогично, изменяя `\vsize` можно управлять вертикальным размером страницы. Начальный TEX устанавливает `\vsize=8.9in` (не `9in`, так как `\vsize` не включает в себя места для номеров страниц внизу каждой страницы). Если вы скажете `\vsize=4in`, то получите более короткие страницы только с четырьмя дюймами текста. Лучше не играть с изменениями `\hsize` и `\vsize`, за исключением самого начала работы или после того, как все страницы выведены из памяти TEX'а.

Если, когда результат в конце концов напечатан, вам хочется, чтобы он был расположен по-другому, можно подвинуть его, задавая ненулевые значения `\hoffset` и `\voffset`. Например,

```
\hoffset=.5in   \voffset=1.5in
```

подвинет результат на полдюйма вправо и на 1.5 дюйма вниз от его нормального положения. Будьте внимательны, чтобы не сдвинуть результат настолько, что он окажется за границей физического устройства, на котором печатается, если только вы не уверены, что такая заграничная деятельность

не приведет к неприятностям.

TEX часто используется для печати объявлений или других бумаг, которым не нужна нумерация страниц. Если вы говорите

```
\nopagenumbers
```

в начале рукописи, начальный TEX воздерживается от вставки номеров внизу каждой страницы.

На самом деле `\nopagenumbers` — это специальный случай более общего механизма, при помощи которого можно управлять верхним и нижним текущим заголовком страницы (верхним и нижним колонтитулами). Программа вывода начального TEX'а помещает специальную строку текста, называемую *headline*, сверху каждой страницы, и другую специальную строку текста, называемую *footline*, снизу. Строка верхнего текущего заголовка (headline) обычно бывает пустой, а в центре строки нижнего текущего заголовка (footline) обычно находится номер страницы. Но можно задать те строки, какие вам хочется, переопределяя команды `\headline` и `\footline`. Например,

```
\headline={\hrulefill}
```

будет помещать горизонтальную черту над каждой страницей. Основная идея в том, что начальный TEX помещает `\line{\the\headline}` вверху страницы, а `\line{\the\footline}` внизу, отделяя эти дополнительные строки от остального материала чистыми строками. (Напомним, что `\line` является аббревиатурой для `\hbox to\hsize`, следовательно, строки верхнего и нижнего заголовка помещаются в боксы такой же ширины, как и обычные строки на самой странице.) Нормальное значение `\headline` равно `\hfil`, так что нет никакого видимого заголовка. Макрокоманда `\nopagenumbers`, описанная ранее — это просто аббревиатура для `\footline={\hfil}`.

Нормальное значение `\footline` равно `\hss\tenrm\folio\hss`. Это центрирует номер страницы на строке, используя шрифт `\tenrm`, поскольку `\folio` — это команда, которая дает номер текущей страницы в текстовой форме.

Номер страницы, как это объяснялось в главе 15, помещается во внутреннем регистре TEX'а `\count0`, и начальный TEX делает `\pageno` аббревиатурой для `\count0`. Таким образом, вы можете сказать `\pageno=100`, если надо, чтобы следующая страница результата имела номер 100. Макрокоманда `\folio` преобразует отрицательные номера страниц в римские числа: если рукопись начинается с `\pageno=-1`, номера страниц будут i, ii, iii, iv, v и т. д. Действительно, приложение B определят `\folio` как аббревиатуру для

```
\ifnum\pageno<0 \romannumeral-\pageno \else\number\pageno \fi
```

Когда вы определяете строки верхнего и нижнего текущего заголовка, важно явно задавать имена шрифтов, поскольку программа вывода TEX'а начинает действовать в некоторое непредсказуемое время. Например, предположим, что `\footline` была установлена в `\hss\folio\hss` без указания `\tenrm`. Тогда номер страницы будет напечатан тем шрифтом, который будет текущем в то

время, когда TEX решит выводить страницу. В таком случае может быть непредсказуемый эффект, поскольку TEX обычно находится в середине страницы 101, когда выводит страницу 100.

▸ **Упражнение 23.1**
Объясните, как в начальном TEX'е поместить en-тире вокруг номеров страниц. Например, внизу страницы 4 должно появиться " – 4 – ".

Приведем пример строки верхнего текущего заголовка, в которой появляется номер страницы. Более того, нечетные и четные страницы обрабатываются по-разному:

```
\nopagenumbers % suppress footlines
\headline={\ifodd\pageno\rightheadline \else\leftheadline\fi}
\def\rightheadline{\tenrm\hfil ПРАВЫЙ ЗАГОЛОВОК\hfil\folio}
\def\leftheadline{\tenrm\folio\hfil ЛЕВЫЙ ЗАГОЛОВОК\hfil}
\voffset=2\baselineskip
```

Англоязычные книги традиционно имеют страницы с нечетными номерами справа, а страницы с четными номерами — слева. Текст, который появляется в строке верхнего текущего заголовка, часто называется "бегущим заголовком". Когда вы используете строки верхнего текущего заголовка, обычно имеет смысл установить `\voffset` как эквивалент двум строкам текста, как показано в нашем примере, так что над страницами результата еще останутся поля в 1 дюйм.

▸ **Упражнение 23.2**
Предположим, что вы используете TEX, чтобы напечатать резюме, которое состоит из нескольких страниц. Объясните, как определить `\headline` так, чтобы первая страница была озаглавлена "**РЕЗЮМЕ**" в центре строки и жирным шрифтом, в то время как каждая последующая строка имела такой бегущий заголовок:

Резюме А. В. Тора ..... Страница 2

Если вы не меняете `\vsize`, все строки верхнего и нижнего текущего заголовка будут появляться на одном и том же месте независимо от содержимого страницы, расположенной между ними. Так, например, если используется `\raggedbottom`, как объяснялось в главе 15, и страницы не всегда содержат одинаковое количество текста, неровности появятся над нижним заголовком; сама строка нижнего текущего заголовка сдвигаться вверх не будет. Если же вы меняете `\vsize`, изменится положение нижнего заголовка, в то время как строка верхнего текущего заголовка будет оставаться на месте.

Оставшаяся часть этой главы предназначена для тех, кто хочет иметь выходной формат, существенно отличающийся от того, который обеспечивает начальный TEX. Во всех следующих абзацах использован двойной знак опасного поворота, поскольку вы должны быть знакомы с остальным TEX'ом до того, как погрузитесь в эти сокровенные тайны языка. Глава 22 научила вас, как стать Мастером TEX'а, т. е. специалистом, который умеет получать сложные таблицы, используя `\halign` и `\valign`. Следующий материал откроет вам все пути к званию Гроссмейстера, т. е. Мастера, который умеет создавать программы вывода. Когда вы подготовитесь к этому званию, вы с удовольствием обнаружите,

что как и в таблицах, в программах вывода на самом деле нет ничего загадочного, как вам в начале могло показаться.

Давайте начнем с повторения некоторых правил, приведенных в конце главы 15. TEX периодически формирует страницу информации для вывода, разбивая свой основной вертикальный список в том месте, которое он считает наилучшим, и в такие моменты он входит во внутреннюю вертикальную моду и начинает читать команды из текущей `\output`-программы. Когда начинается программа вывода, `\box255` содержит страницу, которую закончил TEX; предполагается, что программа вывода делает что-то с этим боксом. Когда программа вывода кончает работу, список элементов, который она создала во внутренней вертикальной моде, помещается прямо перед материалом, который следует за разрывом страницы. При такой работе решение TEX'а о разбиении страницы может быть изменено: какая-то часть или весь материал с разорванной страницы может быть убран и перенесен в начало следующей страницы.

Текущая `\output`-программа определяется как параметр, состоящий из списка элементов точно так же, как `\everypar` или `\errhelp`, за тем исключением, что TEX автоматически вставляет символ начала группы "`{`" в ее начале и символ конца группы "`}`" в конце. Эти символы группирования помогают предохранить программу вывода от перепутывания с деятельностью TEX'а после выбора разрыва страницы. Например, программа вывода, когда помещает на страницу строки верхнего и нижнего текущих заголовков, часто меняет `\baselineskip`; дополнительные фигурные скобки делают это изменение локальным. Если не указано никакой программы `\output` или сказано `\output={}`, TEX применяет свою собственную программу, которая по существу эквивалентна `\output={\shipout\box255}`. Эта программа выводит страницу без строк текущих заголовков и без изменения номера страницы.

Реальный вывод делает примитивная команда TEX'а `\shipout`⟨бокс⟩. Она посылает содержимое бокса в `dvi`-файл, который является основным выходным файлом TEX'а. После того, как TEX закончил работу, `dvi`-файл будет содержать инструкции, закодированные независимо от устройства вывода, которые точно указывают, что должно быть напечатано. Когда бокс выводится, TEX показывает на терминале значения от `\count0` до `\count9`, как объяснялось в главе 15. Эти десять счетчиков тоже записываются в `dvi`-файл, где могут быть использованы для идентификации страницы. Все команды `\openout`, `\closeout` и `\write`, которые встречаются внутри бокса, при передаче бокса выполняются в их естественном порядке. Поскольку команда `\write` раскрывает макроопределения, как объяснялось в главе 21, сканирующий механизм TEX'а во время действия `\shipout` может обнаружить синтаксические ошибки. Если во время `\tracingoutput` значение `\shipout` ненулевое, содержимое выводимого бокса в символической форме записывается в протокольный файл. Можно использовать `\shipout` в любом месте, а не только в программе вывода.

Отложенное выполнение команды `\write` накладывает заслуживающее упоминания ограничение: необходимо гарантировать, чтобы, когда дается команда `\shipout`, все макрокоманды, которые могут появиться внутри текста макрокоманды `\write`, были правильно определены. Например, формат начального TEX'а, описанный в приложении B временно делает пробелы активными

символами и говорит `\global\let␣=\space`. Причина этого в том, что во время выполнения команды `\write` могла действовать команда `\obeyspaces`, так что определение ␣ как активного символа, должно действовать во время работы следующего `\shipout`, даже хотя TEX мог в это время и не считать пробелы активными символами.

Глава 15 отмечает, что TEX дает специальные значения некоторым внутренним регистрам и параметрам в дополнение к `\box255` непосредственно перед тем, как начинает работать программа вывода. Вставки помещаются в их собственные v-боксы, а `\insertpenalties` устанавливается равным общему числу расщепленных вставок. Кроме того, параметр `\outputpenalty` устанавливается равным значению штрафа текущей точки разбиения. Программа вывода может работать специальным образом, когда эти величины имеют специальные значения. Например, программа вывода начального TEX'а распознает `\supereject` (который выводит задержанные вставки) по тому признаку, что `\supereject` указывает `\outputpenalty` быть равным -20000, и использует `\insertpenalties`, чтобы решить, отложена ли некая вставка.

Программа вывода `\shipout\box255`, работающая по умолчанию, иллюстрирует одну крайность, ничего не помещая в вертикальный список, который переносится на следующую страницу. Другая крайность — это

```
\output={\unvbox255 \penalty\outputpenalty}
```

которая ничего не отправляет в dvi-файл и *все* помещает обратно в основной вертикальный список. (Команда `\unvbox255` вынимает завершенную страницу из ее бокса, а команда `\penalty\outputpenalty` перевставляет штраф выбранной точки разбиения.) Это делает гладким соединение между завершенной страницей и последующим материалом, поскольку TEX, когда включает `\output`-программу, еще не имеет отпадающего клея и штрафов на точке разбиения. Следовательно TEX будет возвращаться назад и пересматривать разрыв страницы. Если не изменен `\vsize` и если не было задержанных вставок, будет найден тот же самый разрыв страницы, но намного быстрее, чем прежде, поскольку вертикальный список уже сконструирован и не нужно снова делать абзацы. Конечно, такая программа вывода заставляет TEX крутить свои колеса бесконечно, поэтому она и не используется нигде, кроме примера экстремального случая.

Чтобы предотвратить зацикливание, программа вывода должна каждый раз, когда вступает в дело, делать какой-нибудь шаг вперед. Если вы допустили ошибку, TEX может помочь ее диагностировать, поскольку в него встроен специальный механизм для обнаружения циклов. Имеется внутренняя целая переменная `\deadcycles`, которая обращается в нуль после каждого `\shipout` и увеличивается на 1 перед каждым `\output`. Таким образом, `\deadcycles` сохраняет след того, сколько раз программа вывода была задействована в течение самого последнего `\shipout`, если только вы сами не изменили значение `\deadcycles`. Есть также целый параметр `\maxdeadcycles`, который начальный TEX устанавливает равным 25. Если `\deadcycles` больше или равен `\maxdeadcycles`, когда ваша программа вывода должна начать работать (т. е. когда надо увеличивать `\deadcycles`), TEX сообщает об ошибке и выполняет вместо вашей программы вывода программу вывода по умолчанию.

Когда программа вывода закончена, `\box255` должен быть пустым. Другими словами, вы должны сделать что-нибудь с информацией в этом боксе: она должна быть или выгружена, либо перемещена в какое-нибудь другое место. Аналогично, `\box255` должен быть пустым, когда TEX готовится заполнить его новой страницей материала непосредственно перед началом программы вывода. Если в любой из этих моментов `\box255` не пустой, TEX будет жаловаться, что вы неправильно употребили этот специальный регистр и содержимое регистра будет испорчено.

Но давайте не будем все время говорить о крайних случаях и специальных параметрах, а рассмотрим какие-нибудь реальные примеры. Программа вывода начального TEX, которая приведена в приложении B, задана как `\output={\plainoutput}`, где `\plainoutput` — это аббревиатура для

```
\shipout\vbox{\makeheadline
  \pagebody
  \makefootline}
\advancepageno
\ifnum\outputpenalty>-20000 \else\dosupereject\fi
```

Давайте рассмотрим эту "программу" построчно:

1) Макрокоманда `\makeheadline` создает вертикальный бокс нулевой высоты и глубины таким образом, что строка верхнего текущего заголовка правильно расположена над остальной частью страницы. Вот ее кодировка:

```
\vbox to 0pt{\vskip-22.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}
\nointerlineskip
```

Магическая константа -22.5 pt равна

$$(\text{верхний пропуск} - \text{высота подпорки} - 2(\text{междустрочный пропуск})),$$

т. е. $10\,\text{pt} - 8.5\,\text{pt} - 24\,\text{pt}$. Это помещает точку привязки строки верхнего текущего заголовка в точности на 24 pt выше точки привязки верхней строки страницы, если только строка заголовка или верхняя строка не слишком длинные.

2) Макрокоманда `\pagebody` — это аббревиатура для

```
\vbox to\vsize{\boxmaxdepth=\maxdepth \pagecontents}
```

Значение `\boxmaxdepth` устанавливается в `\maxdepth`, так что v-бокс будет создан в предположениях, что планировщик страниц TEX'а использован для установки `\box255`.

3) Макрокоманда `\pagecontents` производит вертикальный список из всего, что принадлежит телу страницы, а именно, содержания `\box255` вместе с иллю-

страциями (вставленными сверху) и сносками (вставленными снизу):

```
\ifvoid\topins \else\unvbox\topins\fi
\dimen0=\dp255 \unvbox255
\ifvoid\footins\else % имеется сноска
  \vskip\skip\footins
  \footnoterule
  \unvbox\footins\fi
\ifraggedbottom \kern-\dimen0 \vfil \fi
```

Здесь `\topins` и `\footins` — это номера класса вставок для двух видов вставок, которые использует начальный TEX. Если добавляются дополнительные классы вставок, то `\pagecontents` должен быть соответственно изменен. Заметим, что боксы раскрыты, так что клей из вставок может помочь клею на основной странице. Макрокоманда `\footnoterule` в приложении B помещает между страницей и ее сносками разделяющую линию. Это делает чистый вклад в 0 pt в высоту вертикального списка. Регулирование неровного низа достигается вставкой бесконечного клея,который пересиливает растяжимость `\topskip`.

4) Макрокоманда `\makefootline` помещает `\footline` в нужное положение:

```
\baselineskip=24pt
\line{\the\footline}
```

5) Макрокоманда `\advancepageno` обычно увеличивает `\pageno` на $+1$, но если `\pageno` отрицательна (для римских чисел), изменение происходит на -1. Новое значение `\pageno` будет использоваться, когда программа вывода будет вызвана в следующий раз.

```
\ifnum\pageno<0 \global\advance\pageno by-1
\else \global\advance\pageno by 1 \fi
```

6) Наконец, макрокоманда `\dosupereject` создана для очистки всех вставок, которые задержаны, иллюстраций ли, сносок ли, или и тех и других:

```
\ifnum\insertpenalties>0
  \line{} \kern-\topskip \nobreak
  \vfill\supereject\fi
```

Здесь таинственный отрицательный `\kern` удаляет естественную величину клея `\topskip`, который находится над пустой `\line`. Этот пустой бокс-строка не дает `\vfill` исчезнуть при разбиении страницы. Вертикальный список, который получается из `\dosupereject`, помещается в список TEX'а, состоящий из того, что будет выводиться сразу после пересмотра вставок, желающих попасть на страницу, как объяснялось в главе 15. Следовательно, произойдет другой супер-вывод, и процесс будет продолжаться до тех пор, пока не останется никаких вставок.

Упражнение 23.3

Объясните, как изменить программу вывода начального TEX, чтобы она производила вдвое больше страниц. Материал, который обычно шел на страницах 1, 2, 3 и т.д., должен идти на страницах 1, 3, 5, ..., а четные страницы должны быть чистыми, исключая строки верхнего и нижнего текущего заголовка. (Представьте, что на этих чистых страницах позднее будут размещены фотографии.)

Теперь предположим, что желательно иметь двухколоночный формат. Более того, попытаемся изменить начальный TEX так, чтобы он располагал материал в колонках шириной `\hsize=3.2in`. Каждая страница вывода должна содержать две такие колонки, разделенные пробелом в 0.1 in, таким образом, текст на каждой странице должен занимать ширину 6.5 дюйма. Строки колонтитулов должны объединять обе колонки, но вставки в каждой колонке должны быть независимыми, как если бы эти колонки были отдельными страницами книги. Другими словами, каждая колонка должна содержать свои собственные сноски и свои собственные иллюстрации. При этом мы не должны изменять макрокоманду `\pagebody`.

Для решения этой задачи давайте сначала введем новый регистр размера `\fullhsize`, который представляет собой ширину полной страницы.

```
\newdimen\fullhsize
\fullhsize=6.5in \hsize=3.2in
\def\fullline{\hbox to\fullhsize}
```

Макрокоманды `\makeheadline` и `\makefootline` должны быть модифицированы так, чтобы они использовали `\fullline` вместо `\line`.

Новая программа вывода будет использовать команду `\lr`, которая устанавливается равной либо `L`, либо `R`, в зависимости от того, на левой или на правой части страницы помещается следующая колонка. Когда завершена левая колонка, программа вывода просто спасает ее в регистр бокса. Когда завершена правая колонка, программа выводит обе колонки и увеличивает номер страницы.

```
\let\lr=L \newbox\leftcolumn
\output={\if L\lr
    \global\setbox\leftcolumn=\columnbox \global\let\lr=R
  \else \doubleformat \global\let\lr=L\fi
  \ifnum\outputpenalty>-20000 \else\dosupereject\fi}
\def\doubleformat{\shipout\vbox{\makeheadline
    \fullline{\box\leftcolumn\hfil\columnbox}
    \makefootline}
  \advancepageno}
\def\columnbox{\leftline{\pagebody}}
```

Макрокоманда `\columnbox` использует `\leftline`, чтобы гарантировать, что она производит бокс, ширина которого равна `\hsize`. Ширина `\box255` обычно, но не всегда, в начале программы вывода равна `\hsize`. Любая другая ширина испортила бы формат.

Когда заканчивается двухколоночное форматирование, вероятность того, что последняя колонка окажется слева, равна 50%, и в этом случае вывод еще не произойдет. Тогда инструкции

```
\supereject
\if R\lr \null\vfill\eject\fi
```

восполняют пустую правую колонку, гарантируя, что весь накопленный материал будет напечатан. Есть возможность сделать более причудливое взаимное расположение колонок на последней странице, но детали этого очень хитрые, так как надо

также приспособить сноски и другие вставки. Приложение E включает макрокоманды, которые использованы, чтобы сбалансировать колонки в конце указателя из приложения I и начать двухколоночный формат с середины страницы.

Упражнение 23.4
Как надо изменить приведенный выше пример, если надо напечатать что-либо в три колонки ?

Поскольку программа вывода TEX'а запаздывает по отношению к созданию страницы, вы можете получить ошибочные результаты, если необдуманно измените `\headline` или `\footline`. Например, предположим, что вы печатаете книгу, и формат, который вы используете, позволяет главе начинаться с середины страницы. Тогда было бы ошибкой изменить бегущий заголовок в тот момент, когда начинается новая глава, потому что следующая реальная страница вывода могла еще не содержать ничего из новой главы. Рассмотрим также задачу печати словаря или членских списков. Книга с хорошо организованными ссылками показывает текущую область элементов сверху каждой страницы или пары страниц, так что читателю легко пролистывать книгу, когда он ищет отдельные слова или фамилию. Но несинхронный механизм вывода TEX'а делает трудным, если не невозможным, определить, какая область элементов в действительности присутствует на странице.

Поэтому TEX предусматривает способ вводить в список "метки". Эти метки информируют программу вывода об области информации на каждой странице. Общая идея в том, что вы можете сказать

`\mark{`⟨текст метки⟩`}`

в середине той информации, которую набираете, где текст метки — это список элементов, который раскрывается в командах `\edef`, `\message` и т.д. TEX помещает внутреннее представление текста метки в список, который он строит. Затем позднее, когда завершенная страница упакована в `\box255`, TEX позволяет программе вывода ссылаться на первый и последний тексты меток на этой странице.

Чтобы понять это, лучше всего, вероятно, представить, что TEX генерирует произвольно длинный список боксов, клея и других элементов типа штрафов и меток. Каким-то образом этот длинный вертикальный список получается поделенным на страницы и страницы по одной подготовлены к программе вывода. Как только страница помещена в `\box255`, TEX устанавливает значение трех величин, которые действуют, по существу, как макрокоманды:

- `\botmark` — текст метки, который самым последним появился на странице, только что помещенной в бокс;
- `\firstmark` текст метки, который первым появился на странице, только что помещенной в бокс;
- `\topmark` значение, которое имел `\botmark` перед тем, как текущая страница была помещена в бокс.

Перед первой страницей все эти величины нулевые, т. е. раскрываются в ничто. Когда на странице нет меток, они все равны предыдущему `\botmark`.

Например, предположим, что рукопись включает в точности четыре метки и что страницы разбиты так, что `\mark{`α`}` попадает на страницу 2, `\mark{`β`}` и `\mark{`γ`}` — на страницу 4, а `\mark{`δ`}` — на страницу 5. Тогда

На странице	`\topmark`	`\firstmark`	`\botmark`
1	нулевая	нулевая	нулевая
2	нулевая	α	α
3	α	α	α
4	α	β	γ
5	γ	δ	δ
6	δ	δ	δ

Когда вы используете команду `\mark` в вертикальной моде, TEX помещает метку в основной вертикальный список. Когда вы используете команду `\mark` в горизонтальной моде, TEX рассматривает ее как материал вертикальной моды типа `\vadjust` и `\insert`, т. е. после того, как абзац разбит на строки, каждая метка будет занесена в основной вертикальный список сразу после бокса для строки, в которой эта метка первоначально появилась. Если вы используете `\mark` в ограниченной горизонтальной моде, метка может быть передана в охватывающий вертикальный список таким же образом, как это делают `\insert` и `\vadjust` (см. главу 24). Но метка, которая спрятана слишком глубоко внутри бокса, не будет передана, поэтому никогда не появится в качестве `\firstmark` или `\botmark`. Аналогично, `\mark`, которая встречается во внутренней вертикальной моде, отправляется в v-бокс и является недоступной в основном вертикальном списке.

Глава 15 обсуждает команду `\vsplit`, которая позволяет вам самостоятельно разрывать вертикальные списки. Эта операция иногда обеспечивает полезную альтернативу обычному механизму TEX'а по построению страниц. Например, если вы просто хотите напечатать некоторый материал в двух колонках равной высоты, то можете поместить этот материал в v-бокс, затем с помощью `\vsplit` расщепить бокс на два куска — и не надо никакой программы вывода. Операция `\vsplit` устанавливает значения двух макро-подобных величин, которые не упоминались в главе 15: `\splitfirstmark` и `\splitbotmark` раскрываются в тексты первой и последней меток, которые появляются в вертикальном списке, расщепленном при помощи самой последней команды `\vsplit`. Обе величины нулевые, если не было таких меток. Значения `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark` и `\splitbotmark` являются глобальными, т. е. на них не действует механизм группирования TEX'а.

При печати большинства словарей используется эквивалентность меток `\firstmark` и `\botmark`, чтобы задать путеводные слова вверху каждой пары развернутых страниц. Например, если определение слова "type" начинается на странице 1387 и продолжается на странице 1388, путеводное слово на странице 1387 (правой странице) будет "type"; а на странице 1388 (левой странице) путеводным будет следующее слово словаря (например, "typecast"), даже хотя верх страницы 1388 занято продолжением слова "type".

Словарная схема прекрасно работает для словарей, поскольку читатель должен читать каждый элемент с его начала. Но для технических книг, таких как книга автора *Искусство Программирования*, подходит другая схема, где

секция 1.2.8 (например) начинается в середине страницы 78, но верхняя часть страницы 78 содержит упражнение 19–24 из секции 1.2.7. Заглавная строка наверху страницы 78 указывает "1.2.7", поскольку это поможет тем, кто ищет упражнение 1.2.7–22. Заметим, что соглашение, принятое для словарей, поставило бы "1.2.8" вверху страницы 78, но это удобно, только если секция 1.2.8 начинается с самого верха этой страницы.

Продолжая пример из *Искусства Программирования*, давайте предположим, что рукопись TEX для секции 1.2.8 начинается с макрокоманды

```
\beginsection 1.2.8. Числа Фибоначчи.
```

Как должна быть определена `\beginsection`? Сделаем первую попытку:

```
\def\beginsection #1. #2.
  {\sectionbreak
  \leftline{\sectionfont #1. #2}
  \mark{#1}
  \nobreak\smallskip\noindent}
```

Макрокоманда `\sectionbreak` должна склонять TEX либо прервать страницу на текущей позиции, либо оставить пробел подходящей величины. Например, она может быть аббревиатурой для `\penalty-200 \vskip18pt plus4pt minus6pt`. Макрокоманда `\beginsection` оканчивается командами, которые подавляют отступ первого абзаца секции. Но нас интересует команда `\mark`, которая следует за `\leftline`. В примере, который мы рассматриваем, начало секции 1.2.8 вставило бы `\mark{1.2.8}` в основной вертикальный список сразу после бокса, который содержит заголовок этой секции.

Существует ли такая подходящая `\mark`? К сожалению нет, даже если мы для простоты допустим, что на каждой странице начинается не больше одной секции. Страница, которая содержит начало секции 1.2.8, будет тогда иметь `\topmark=1.2.7` и `\firstmark=1.2.8` безотносительно к тому, начинается или нет секция в самом начале страницы. То, что нам надо — это гибрид между `\topmark` и `\firstmark`, нечто, что будет отражать текст метки, который предоставляет состояние дел сразу после первой строки страницы. А TEX не предусматривает такое.

Решением будет задать `\mark` непосредственно перед `\sectionbreak`, вместо того, чтобы сделать это после `\leftline`. Тогда `\topmark` будет всегда отражать истину о секции, которая является текущей на верхней строке. (Подумайте над этим.)

Однако формат для *Искусства Программирования* еще более сложен. На левых страницах номер секции в бегущем заголовке должен отражать ситуацию вверху страницы, как мы только что обсудили, а на правых страницах он отсылает к нижней части страницы. Наше решение сделало правильную `\topmark` для верхушки, но может делать неправильную `\botmark` для низа. Для того, чтобы выполнить оба требования, необходимо в метки упаковать больше информации.

Приведем один из способов решения этой проблемы:

```
\def\beginsection #1. #2.
  {\mark{\currentsection \noexpand\else #1}
  \sectionbreak
  \leftline{\sectionfont #1. #2}
  \mark{#1\noexpand\else #1} \def\currentsection{#1}
  \nobreak\smallskip\noindent}
\def\currentsection{} % номер текущей секции
```

Идея заключается в том, чтобы ввести две метки, одну перед разрывом секции и одну после того, как секция началась. Более того, каждая метка состоит из двух частей: метка перед потенциальным разрывом между секциями 1.2.7 и 1.2.8 равна `1.2.7\else 1.2.8`, а метка после этого потенциального разрыва равна `1.2.8\else 1.2.8`. Отсюда следует, что номером секции соответствующей нижней части страницы является левая компонента `\botmark`, а номером секции соответствующей верхней части страницы является правая компонента `\topmark`. Макрокоманда `\rightheadline` может использовать `\iftrue\botmark\fi`, чтобы читать левую компоненту, а для чтения правой компоненты `\leftheadline` может сказать `\expandafter\iffalse\topmark\fi`.

▸ **Упражнение 23.5**
Т. У. Пицын использовал конструкцию, очень похожую на описанную выше, но поставил вторую `\mark` перед `\leftline`, а не после нее. Что получилось неправильно?

▸ **Упражнение 23.6**
Метки в предыдущей конструкции имеют форму α`\else`β, где α и β — две независимых части информации. `\else` дает возможность выбрать α или β по значению `\iftrue` и `\iffalse`. Обобщим эту идею. Предположим, у вас есть приложение в котором метки должны содержать пять независимых частей информации, и что каждая метка имеет вид α_0`\or`α_1`\or`α_2`\or`α_3`\or`α_4. Объясните, как из такой метки выбрать каждую из пяти α.

Давайте завершим обсуждение программ вывода, рассматривая составление указателей типа указателя к этому руководству, который приведен в приложении I. Наиболее сложные элементы в таком указателе будут выглядеть так:

Основной элемент, 4, 6, 8–10, 12, 14–16,
18–22, 24–28, 30.
первый вспомогательный элемент, 1–3, 6, 10–11,
15, 21, 24.
второй вспомогательный элемент, 1, 3, 6–7,
10, 15, 21, 25, 28, 31.

Основные и вспомогательные элементы набраны с неровным правым краем и с подвешенным отступом, равным два em, после первой строки. У вспомогательных элементов на первой строке сделан отступ в один em. Нам надо, чтобы такой

результат получался, если мы вводим следующее:

```
\beginindex
...
Основной элемент, 4, 6, 8--10, 12, 14--16, 18--22, 24--28, 30.
\sub первый вспомогательный элемент, 1--3, 6, 10--11, 15, 21, 24.
\sub второй вспомогательный элемент, 1, 3, 6--7, 10, 15, 21, 25, %
  28, 31.
...
\endindex
```

где "..." обозначает другие элементы. Каждая строка входа обычно задает один основной элемент или один вспомогательный элемент. Если элемент настолько длинный, что не помещается на одной входной строке, то в конце строки вводится "␣%", так что она сливается со своим продолжением.

Интересным в проблеме указателя является то, что желательно установить такую систему меток, чтобы программа вывода вставляла специальные строки текста, когда элемент разрывается между колонками или страницами. Например, если разрыв страницы приходится между любыми шестью строками напечатанного вывода, который показан выше, программа вывода должна выдать специальную строку

Основной элемент (*продолжение*):

а если разрыв страницы оказывается внутри вспомогательного элемента, то должна появиться специальная дополнительная строка

вспомогательный элемент (*продолжение*):

Приведенное ниже решение обеспечивает такие метки, что `\botmark` будет нулевой, если разрыв произойдет между основными элементами; она будет равна "`Основной элемент`", если разрыв произойдет после строк 1, 2 или 4 в шестистрочном примере вывода; она будет равна "`Основной элемент\sub первый вспомогательный элемент`", если разрыв произойдет после строки 3 (внутри первого вспомогательного элемента); и она будет "`Основной элемент\sub второй вспомогательный элемент`", если разрыв произойдет после строки 5.

Читатель может захотеть попытаться решить эту проблему перед тем, как посмотреть на решение, поскольку тогда будет легче оценить тонкие выводы, которые отсюда следуют. (Приступайте и попробуйте определить макрокоманду `\beginindex`, которая устанавливает неровный правый край и делает указанные метки. Вернитесь на предыдущую страницу, чтобы внимательно изучить проблему, прежде чем заглянуть в ответ.)

```
\def\beginindex{\begingroup
  \parindent=1em \maxdepth=\maxdimen
  \def\par{\endgraf \futurelet\next\inxentry}
  \obeylines \everypar={\hangindent 2\parindent}
  \exhyphenpenalty=10000 \raggedright}
```

```
\def\inxentry{\ifx\next\sub \let\next=\subentry
  \else\ifx\next\endindex \let\next=\vfill
  \else\let\next=\mainentry \fi\fi \next}
\def\endindex{\mark{}\break\endgroup}
\let\sub=\indent \newtoks\maintoks \newtoks\subtoks
\def\mainentry#1,{\mark{}\noindent
  \maintoks={#1}\mark{\the\maintoks}#1,}
\def\subentry\sub#1,{\mark{\the\maintoks}\indent
  \subtoks={#1}\mark{\the\maintoks\sub\the\subtoks}#1,}
```

Даже если вы прочли это решение, вам, вероятно, нужны объяснения, поскольку здесь использовались "TEXовинки", которые в этом руководстве еще не появлялись.

1) Макрокоманда `\beginindex` использует `\begingroup`, чтобы сделать другие изменения локальными. Поэтому не надо будет восстанавливать `\parindent`, `\maxdepth` и т. д. в их прежние значения, когда указатель закончится. Параметр `\maxdepth` устанавливается в `\maxdimen`, который по существу бесконечен, так что `\box255` будет иметь истинную глубину последнего бокса, который в нем содержится; ниже мы будем использовать этот факт. (Можно спокойно портить `\maxdepth` таким способом, поскольку предполагается, что элементы указателя имеют умеренно маленькую глубину.) Заметим, что используется `\obeylines`, так что в конце каждой строки ввода будет вставляться `\par`. Значение `\par` изменено, так что он делает больше, чем обычно. Сначала он выполняет `\endgraf`, который является обычной TEXовской операцией `\par`, затем помещает `\next` в первый элемент следующей строки, после чего будет раскрываться макрокоманда `\inxentry`.

2) Когда вступает в дело `\inxentry`, она смотрит на `\next`, чтобы решить, что делать. Могут быть три случая. Если `\next` равно `\sub`, строка будет трактоваться как вспомогательный элемент. Если `\next` равно `\endindex`, то следующей выполняемой командой будет `\vfill\mark{}\break\endgroup`. В других случаях строка будет рассматриваться как основной элемент.

3) Текст основного элемента помещается в параметр `#1` команды `\mainentry`. Этот параметр ограничивается запятой. Сначала `\mainentry` выполняет `\mark{}`, которая очищает метку при разрыве между элементами. Затем идет `\noindent`, которая указывает TEX'у перейти в горизонтальную моду и выделить `\parskip`-клей. (Клей `\parskip` будет разрешенной точкой разрыва между строками. За ним позже будет следовать междустрочный клей, когда первая строка основного элемента будет напечатана при помощи абзацной программы TEX'а.) Затем другая `\mark` помещается в сам абзац. Она содержит текст основного элемента, а `\toks`-регистр с именем `\maintoks` используется, чтобы задержать раскрытие текста метки. Когда абзац завершен и поделен на строки, эта особая метка будет следовать сразу боксом для первой строки абзаца, так что она будет равна `\botmark`, если где-нибудь внутри абзаца случится разрыв.

4) Аналогичная конструкция используется для `\subentry`, но метка здесь более сложная. Регистр `\maintoks` будет содержать еще и основной элемент. Текст для вспомогательного элемента добавляется с помощью другого регистра списка элементов, `\subtoks`. Поскольку `\sub` определен равным `\indent`, он не будет раскрыт в эту `\mark`.


Макрокоманды, которые только что определены, будут печатать элементы, которые содержат необходимые метки. Теперь мы должны сконструировать программу вывода, которая использует эти метки, чтобы вставить, как упомянуто выше, новые строки с текстом "(*продолжение*)". И снова советуем читателю попытаться решить эту проблему, прежде чем посмотреть на решение.

```
\output={\dimen0=\dp255 \normaloutput
  \expandafter\inxcheck\botmark\sub\end}
\def\inxcheck#1\sub#2\end{\def\next{#1}%
  \ifx\next\empty % ничего не делает, если \botmark нулевой
  \else\noindent #1\continued % ``Основной элемент (продолжение):''
    \def\next{#2}%
    \ifx\next\empty % ничего более, если \botmark не имеет \sub
    \else\let\sub=\continued \indent #2\fi
    \advance\dimen0 by-\prevdepth \kern\dimen0 \fi}
\def\continued{ ({\it продолжение}\thinspace):\endgraf}
```

Эта программа несколько более хитроумна, чем обычно. Она предполагает, что `\normaloutput` занимается выводом `\box255` (возможно, помещая его в многоколоночный формат) и изменением номера страницы. Затем следует новый материал, который выполняется при помощи `\inxcheck`. Макрокоманда `\inxcheck` вызывается интересным способом, который позволяет `\botmark` разделиться на ее компоненты. Если `\botmark` нулевая, то аргумент `#1` команды `\inxcheck` будет нулевым, следовательно, `\next` будет эквивалентом `\empty`. (Для того, чтобы уладить такие ситуации, начальный TEX говорит `\def\empty{}`.) Если `\botmark` не содержит элемента `\sub`, то содержанием аргумента `#1` будет `\botmark`, в то время как `#2` будет нулевым. В противном случае, если `\botmark` имеет вид α`\sub`β, аргументом `#1` будет α, а аргументом `#2` будет β`\sub`.


Если `\botmark` не нулевая, макрокоманда `\inxcheck` производит одну или более строк текста, которые будут занесены в основной вертикальный список TEX'а на место разрыва страницы, вычисленное на основании глубины бокса, который предшествовал разрыву. На разрыве странице будет вставлен междустрочный клей, вычисленный на основании глубины бокса, предшествовавшего разрыву. Эта глубина известна программе вывода, поскольку она является глубиной `\box255`. (Значение `\maxdepth` было сделано бесконечным по той же причине.) Поэтому макрокоманда `\inxcheck` может вставить `\kern`, чтобы компенсировать разницу в глубинах между старым боксом и тем боксом, который будет вставлен перед уже вычисленным междустрочным клеем. Без этого `\kern` распределение пробелов будет неправильным. Читатель должен внимательно изучить этот пример, чтобы понять рассуждения по поводу команды `\kern`, прежде чем создавать программу вывода, которая наугад вставляет новые боксы между строками вывода.


▸ **Упражнение 23.7**
Измените эту конструкцию так, чтобы строки продолжения вставлялись только в левых колонках страниц с четными номерами, предполагая двухколочный формат.


▸ **Упражнение 23.8**
Правильно или нет следующее утверждение: макрокоманда `\inxcheck` в

этом примере добавляет не более двух строк вывода в основной вертикальный список.

Когда TEX видит команду `\end`, он оканчивает работу, только если основной вертикальный список полностью выведен и если `\deadcycles=0`. В противном случае он вставляет в основной вертикальный список эквивалент для

```
\line{} \vfill \penalty-'10000000000
```

и приготавливается читать `\end` снова. Это приводит к повторным вызовам программы вывода до тех пор, пока не будет выведено все. В частности, последняя колонка двухколоночного формата не будет утеряна.

Можно придумать такие программы вывода, которые всегда оставляют остаток в основном вертикальном списке и к тому же не позволяют `\deadcycles` увеличиваться. В таком случае TEX никогда не дойдет до конца! Однако программа вывода может узнать, что она вызвана Эндшпилем TEX'а, поскольку имеется специальный `\outputpenalty`, который задает большой отрицательный `\penalty-'10000000000`. В таких случаях программа вывода должна изменить свое поведение, если это необходимо, что гарантирует счастливый конец.

Я думаю, они тебе понравятся,
Когда ты их увидишь
на прекрасной просторной странице,
где изящный ручеек текста
будет виться
по лужайке полей.
Ей-богу, они будут самыми элегантными
вещицами в этом роде!

I think you will like them,
when you shall see them
on a beautiful quarto page,
where a neat rivulet of text
shall meander through
a meadow of margin.
'Fore Gad they will be the most
elegant things of their kind!

— RICHARD BRINSLEY SHERIDAN, *The School for Scandal* (1777)

Влияние технических изменений
на производительность при вариациях
общего уровня инвестиций β
так мало, что на деле
им можно пренебречь.

The influence of technical changes
upon outputs through variation
in the general investment level β
is so small that actually
it could have been neglected.

— WASSILY W. LEONTIEF, *The Structure of American Economy, 1919–1929* (1941)

24

Обзор вертикальной моды

В предыдущих главах представлен весь язык TEX'а. Наконец мы достигли конца нашего путешествия по территории, не нанесенной на карту. Ура! Победа! Теперь настало время более систематично посмотреть на то, с чем мы сталкивались, рассмотреть факты по порядку, не перемешивая их с неформальными примерами и приложениями, как мы это делали до этого. Ребенок учится говорить до того, как выучит формальные правила грамматики, но правила грамматики оказываются кстати, когда он достигает совершеннолетия. Цель этой главы — и двух следующих за ней глав — дать точный и сжатый обзор того языка, который понимает TEX, так что подготовленные пользователи смогут наиболее эффективно общаться с машиной.

В этих главах мы будем интересоваться только *примитивными* операциями TEX'а, а не элементами высокого уровня формата начального TEX'а, с которыми имеет дело большинство людей. Поэтому начинающий пользователь должен отложить чтение глав 24–26 до тех пор, пока не почувствует необходимость узнать, что происходит внутри компьютера. Приложение B содержит обзор начального TEX'а вместе с ссылками на то, что большинство людей хотят знать об использовании TEX'а. Чтобы получить обзор TEX'а высокого уровня, лучше всего обратиться к изучению приложения B.

Нашей целью здесь, тем не менее, является сделать обзор команд низкого уровня, на которых построены суперструктуры высокого уровня, для того, чтобы обеспечить подробные ссылки тем, кому надо знать подробности. Остаток этой главы напечатан более мелким шрифтом, как и этот абзац, поскольку аналогичный материал в других главах помечался "двойными знаками опасного поворота". Вместо того, чтобы использовать повторение знаков опасного поворота, давайте просто условимся, что главы 24–26 опасны по определению.

TEX в действительности имеет некоторые возможности, которые не удостоились упоминания в предыдущих главах, поэтому будут введены здесь как часть полного обзора. В случае какой-либо несогласованности между тем, что было сказано раньше и будет сказано впереди, факты из этой и последующих глав считаются лучшим приближением к истине.

Мы будем изучать пищеварительные процессы TEX'а, то есть действия TEX'а со списками элементов, которые поступают в его "желудок". В главе 7 описан процесс, при котором входные файлы преобразуются в списки элементов в "пасти", TEX'а, а глава 20 объясняла, как раскрываемые элементы преобразуются в нераскрываемые в "глотке" TEX'а с помощью процесса, похожего на отрыжку. Когда нераскрываемые элементы наконец достигают желудочнокишечного тракта TEX'а, начинается реальная деятельность по набору, и это именно то, о чем мы собираемся говорить в этих итоговых главах.

Каждый элемент, который поступает в желудок TEX'а, рассматривается как *команда*, которой будет повиноваться компьютер. Например, буква `L` — это команда напечатать L в текущем шрифте, а `\par` говорит TEX'у закончить абзац. TEX всегда находится в одной из шести мод, как это описано в главе 13, и команды иногда в разных модах означают разные вещи. Эта глава посвящена вертикальной моде (и внутренней вертикальной моде, что почти то же самое). Мы обсудим, как

TEX реагирует на каждую примитивную команду, когда эта команда встречается в вертикальной моде. Главы 25 и 26, аналогично, характеризуют горизонтальную и математическую моды, но те главы короче этой, поскольку многие команды во всех модах ведут себя одинаково. Правила для таких команд не будут повторяться трижды, они появятся только один раз.

Некоторые команды имеют аргументы. Другими словами, один или более элементов, которые следуют за командой, могут быть использованы, чтобы изменить поведение этой команды, а сами эти элементы не рассматриваются как команды. Например, когда TEX обрабатывает последовательность элементов, которая соответствует `\dimen2=2.5pt`, он считает командой только первый элемент `\dimen`. Следующие элементы просматриваются при выполнении операции, поскольку TEX'у надо знать, какой регистр `\dimen` устанавливается равным какому значению размера.

Мы будем определять части речи TEX'а, используя модифицированную форму грамматической записи, которая была введена около 1960 года Джоном Бэкусом и Петером Нором для определения компьютерных языков. Величины в угловых скобках будут объяснены словами, либо определены по *синтаксическим правилам*, которые точно показывают, как они формируются из других величин. Например,

⟨единица измерения⟩ ⟶ ⟨возможные пробелы⟩⟨внутренняя единица⟩
| ⟨возможное `true`⟩⟨физическая единица⟩

определяет, что ⟨единица измерения⟩ — это либо ⟨возможный пробел⟩, за которым следует ⟨внутренняя единица⟩, либо ⟨возможное `true`⟩ за которым следует ⟨физическая единица⟩. Обозначение ⟶ означает "определено как", а | означает "или".

Иногда синтаксические правила являются рекурсивными в том смысле, что правая часть определения содержит величину, которая им определяется. Например, правило

⟨возможные пробелы⟩ ⟶ ⟨пусто⟩
| ⟨элемент пробела⟩⟨возможные пробелы⟩

говорит, что грамматическая величина ⟨возможные пробелы⟩ — это либо ⟨пусто⟩, либо ⟨элемент пробела⟩, за которым следуют ⟨возможные пробелы⟩. Величина ⟨пусто⟩ обозначает "ничего", то есть вообще нет элемента. Следовательно, это синтаксическое правило формализует способ сказать, что ⟨возможные пробелы⟩ — это последовательность из нуля или более пробелов.

Альтернативы в правой части синтаксического правила не обязательно должны состоять целиком из величин в угловых скобках. Могут быть также использованы и элементы в явной форме. Например, правило

⟨плюс или минус⟩ ⟶ `+`$_{12}$ | `-`$_{12}$

говорит, что ⟨плюс или минус⟩ обозначает символьный элемент, равный либо знаку плюс, либо знаку минус с номером категории 12.

Мы будем использовать специальное соглашение для ключевых слов, потому что действительный синтаксис ключевых слов — это нечто очень техническое. Буквы шрифта пишущей машинки типа "`pt`" будут обозначать

⟨возможные пробелы⟩⟨p или P⟩⟨t или T⟩,

где ⟨p или P⟩ обозначает некоторый неактивный символьный элемент для `p` или `P` (независимый от номера категории), а ⟨t или T⟩ — аналогично.

Когда команда типа `\dimen` используется в синтаксических правилах, она обозначает некоторый элемент, текущее значение которого такое же, какое имел `\dimen`, когда TEX стартовал. Другим элементам может быть присвоено это же самое значение, используя `\let` или `\futurelet`, а значение самой команды `\dimen` может быть переопределено пользователем, но синтаксические правила не содержат записи этого. Они используют `\dimen` только как способ ссылаться на отдельную примитивную команду TEX'а. (Эта запись отличается от `dimen`, которая обозначает элемент-команду, имя которой "`dimen`"; см. главу 7.)

Команды иногда маскируются под символы, если значение им присвоено при помощи `\let` или `\futurelet`. Например, приложение B говорит

```
\let\bgroup={   \let\egroup=}
```

и эти команды делают так, что `\bgroup` и `\egroup` действуют, как левая и правая фигурные скобки. Такие команды называются "неявными символами". Они интерпретируются как символы, когда TEX исполняет их как команды, но не всегда, если они появляются как аргументы команд. Например, команда `\let\plus=+` не делает `\plus` допустимой заменой для символьного элемента "$+_{12}$" в синтаксическом правиле для ⟨плюс или минус⟩, как и команда `\let\p=p` не делает `\p` приемлемым в ключевом слове `pt`. В тех случаях, когда синтаксис TEX'а позволяет применять как явные, так и неявные символы, правила, приведенные ниже, будут четко и явно говорить об этом.

Величина ⟨знак пробела⟩, которая была использована в синтаксисе для ⟨возможные пробелы⟩, обозначает явный и неявный пробел. Другими словами, она обозначает либо символьный элемент категории 10, либо команду или активный символ, текущее значение которого сделано равным такому элементу при помощи `\let` или `\futurelet`.

Будет удобно использовать обозначения `{`, `}` и `$` для указания явных или неявных символьных элементов, соответственно, категорий 1, 2 и 3, независимо от того, имеют они своими действительными символьными кодами фигурные скобки или знаки доллара или нет. Так, например, `\bgroup` начального TEX'а является примером `{`, а также $\texttt{\{}_1$ и $\texttt{(}_1$, но не $\texttt{\{}_{12}$.

Из нескольких последних абзацев следует, что альтернативы в правых частях формальных синтаксических правил TEX'а состоят из одной или более следующих вещей: (1) синтаксические величины типа ⟨возможные пробелы⟩, (2) явные символьные элементы типа $+_{12}$, (3) ключевые слова типа `pt`, (4) имена команд типа `\dimen` или (5) специальные обозначения `{`, `}` и `$`.

Давайте начнем наше изучение синтаксиса TEX'а с обсуждения точного значения таких величин, как ⟨число⟩, ⟨размер⟩ и ⟨клей⟩, которые часто встречаются в качестве аргументов команд. Наиболее важным из них является ⟨число⟩, которое задает целое значение. Укажем точно, что же такое ⟨число⟩:

⟨число⟩ ⟶ ⟨возможные знаки⟩⟨число без знака⟩
⟨возможные знаки⟩ ⟶ ⟨возможные пробелы⟩
  | ⟨возможные знаки⟩⟨плюс или минус⟩⟨возможные пробелы⟩
⟨число без знака⟩ ⟶ ⟨нормальное целое⟩ | ⟨приведенное целое⟩

⟨нормальное целое⟩ ⟶ ⟨внутреннее целое⟩
| ⟨целая константа⟩⟨один возможный пробел⟩
| ’$_{12}$⟨восьмеричная константа⟩⟨один возможный пробел⟩
| "$_{12}$⟨шестнадцатеричная константа⟩⟨один возможный пробел⟩
| ‘$_{12}$⟨символьный элемент⟩⟨один возможный пробел⟩
⟨целая константа⟩ ⟶ ⟨цифра⟩ | ⟨цифра⟩⟨целая константа⟩
⟨восьмеричная константа⟩ ⟶ ⟨восьмеричная цифра⟩
| ⟨восьмеричная цифра⟩⟨восьмеричная константа⟩
⟨шестнадцатеричная константа⟩ ⟶ ⟨шестнадцатеричная цифра⟩
| ⟨шестнадцатиричная цифра⟩⟨шестнадцатиричная константа⟩
⟨восьмеричная цифра⟩ ⟶ $\texttt{0}_{12}$ | $\texttt{1}_{12}$ | $\texttt{2}_{12}$ | $\texttt{3}_{12}$ | $\texttt{4}_{12}$ | $\texttt{5}_{12}$ | $\texttt{6}_{12}$ | $\texttt{7}_{12}$
⟨цифра⟩ ⟶ ⟨восьмеричная цифра⟩ | $\texttt{8}_{12}$ | $\texttt{9}_{12}$
⟨шестнадцатиричная⟩ ⟶ ⟨цифра⟩ | $\texttt{A}_{11}$ | $\texttt{B}_{11}$ | $\texttt{C}_{11}$ | $\texttt{D}_{11}$ | $\texttt{E}_{11}$ | $\texttt{F}_{11}$
| $\texttt{A}_{12}$ | $\texttt{B}_{12}$ | $\texttt{C}_{12}$ | $\texttt{D}_{12}$ | $\texttt{E}_{12}$ | $\texttt{F}_{12}$
⟨один возможный пробел⟩ ⟶ ⟨элемент пробела⟩ | ⟨пусто⟩
⟨приведенное целое⟩ ⟶ ⟨внутренний размер⟩ | ⟨внутренний клей⟩

⟨Число⟩ — это соответствующее ⟨число без знака⟩, умноженного на -1 для каждого знака минус в ⟨возможные знаки⟩. Алфавитная константа означает символьный код в ⟨символьный элемент⟩; TEX не раскрывает этот элемент, который должен быть либо парой (символьный код, номер категории), либо активным символом, либо командой, имя которой состоит из одного символа. (См. в главе 20 полный список всех ситуаций, в которых TEX не раскрывает элементы.) После ⟨целая константа⟩ не должна сразу следовать ⟨цифра⟩. Другими словами, если несколько цифр появляются последовательно, они все рассматриваются как часть одной и той же целой константы. Аналогичное замечание относится к таким величинам, как ⟨восьмеричная константа⟩ и ⟨шестнадцатеричная константа⟩. Величина ⟨один возможный пробел⟩ равна ⟨пусто⟩, только если он должен быть, то есть TEX ищет ⟨один возможный пробел⟩, читая элемент и возвращаясь, если его там нет.

Упражнение 24.1
Можете ли вы придумать причину, почему можно захотеть, чтобы $\texttt{A}_{12}$ было бы шестнадцатеричной цифрой, даже хотя буква `A` имеет категорию 11? (Не волнуйтесь, если отвечаете “нет”.)

Чтобы закончить определение числа, осталось дать определение трем величинам: ⟨внутреннее целое число⟩, ⟨внутренний размер⟩ и ⟨внутренний клей⟩, которые будут объяснены позже. Они представляют собой вещи типа параметров или регистров. Например, `\count1`, `\tolerance` и `\hyphenchar\tenrm` — это внутренние целые, `\dimen10`, `\hsize` и `\fontdimen6\tenrm` — внутренние размеры, `\skip100`, `\baselineskip` и `\lastskip` — внутренний клей. Внутренние размеры могут быть “приведены” к целому числу в единицах суперпунктов. Например, если `\hsize=100pt` и если `\hsize` используется в контексте, как ⟨число⟩, оно обозначает целую величину 6553600. Аналогично, внутренний клей может быть приведен к целому числу — сначала сделан размером (опуская растяжимость и сжимаемость), затем размер приведен к целому числу.

Давайте теперь вернемся к синтаксису для величины ⟨размер⟩ и ее родственника ⟨mu-размер⟩:

⟨размер⟩ ⟶ ⟨возможные знаки⟩⟨размер без знака⟩
⟨размер без знака⟩ ⟶ ⟨нормальный размер⟩ | ⟨приведенный размер⟩
⟨приведенный размер⟩ ⟶ ⟨внутренний клей⟩
⟨нормальный размер⟩ ⟶ ⟨внутренний размер⟩
| ⟨коэффициент⟩⟨единица измерения⟩
⟨коэффициент⟩ ⟶ ⟨нормальное целое⟩ | ⟨десятичная константа⟩
⟨десятичная константа⟩ ⟶ `.`12 | `,`12
| ⟨цифра⟩⟨десятичная константа⟩
| ⟨десятичная константа⟩⟨цифра⟩
⟨единица измерения⟩ ⟶ ⟨возможные пробелы⟩⟨внутренняя единица⟩
| ⟨возможное `true`⟩⟨физическая единица⟩⟨один возможный пробел⟩
⟨внутренняя единица⟩ ⟶ `em`⟨один возможный пробел⟩
| `ex`⟨один возможный пробел⟩
| ⟨внутреннее целое⟩ | ⟨внутренний размер⟩ | ⟨внутренний клей⟩
⟨возможный `true`⟩ ⟶ `true` | ⟨пусто⟩
⟨физическая единица⟩ ⟶ `pt` | `pc` | `in` | `bp` | `cm` | `mm` | `dd` | `cc` | `sp`

⟨mu-размер⟩ ⟶ ⟨возможные знаки⟩⟨mu-размер без знака⟩
⟨mu-размер без знака⟩ ⟶ ⟨нормальный mu-размер⟩
| ⟨приведенный mu-размер⟩
⟨приведенный mu-размер⟩ ⟶ ⟨внутренний mu-клей⟩
⟨нормальный mu-размер⟩ ⟶ ⟨коэффициент⟩⟨mu-единица⟩
⟨mu-единица⟩ ⟶ ⟨возможные пробелы⟩⟨внутренний mu-клей⟩
| `mu`⟨один возможный пробел⟩

Когда присутствует `true`, коэффициент умножается на 1000 и делится на параметр `\mag`. Физические единицы определяются в главе 10, `mu` объясняется в главе 18.

Ободренные нашими успехами в создании точного синтаксиса для величин ⟨число⟩, ⟨размер⟩ и ⟨mu-размер⟩, давайте добавим ⟨клей⟩ и ⟨mu-клей⟩:

⟨клей⟩ ⟶ ⟨возможные знаки⟩⟨внутренний клей⟩
| ⟨размер⟩⟨растяжимость⟩⟨сжимаемость⟩
⟨растяжимость⟩ ⟶ `plus`⟨размер⟩ | `plus`⟨fil размер⟩
| ⟨возможные пробелы⟩
⟨сжимаемость⟩ ⟶ `minus`⟨размер⟩ | `minus`⟨fil размер⟩
| ⟨возможные пробелы⟩
⟨fil размер⟩ ⟶ ⟨возможные знаки⟩⟨коэффициент⟩
⟨fil единица⟩⟨возможные пробелы⟩
⟨fil единица⟩ ⟶ `fil` | ⟨fil единица⟩`l`
⟨mu-клей⟩ ⟶ ⟨возможные знаки⟩⟨внутренний mu-клей⟩
| ⟨mu-размер⟩⟨mu-растяжимость⟩⟨mu-сжимаемость⟩
⟨mu-растяжимость⟩ ⟶ `plus`⟨mu-размер⟩ | `plus`⟨fil размер⟩
| ⟨возможные пробелы⟩
⟨mu-сжимаемость⟩ ⟶ `minus`⟨mu-размер⟩ | `minus`⟨fil размер⟩
| ⟨возможные пробелы⟩

TEX допускает большое количество внутренних величин, так что разработчик формата может влиять на поведение TEX'а. Приведем список этих величин, за исключением параметров (которые будут перечислены позже).

⟨внутреннее целое⟩ ⟶ ⟨целый параметр⟩ | ⟨специальное целое⟩
| `\lastpenalty`
| ⟨countdef-элемент⟩ | `\count`⟨8-битное число⟩
| ⟨chardef-элемент⟩ | ⟨mathchardef-элемент⟩
| `\hyphenchar`⟨шрифт⟩ | `\skewchar`⟨шрифт⟩
| ⟨имя кода⟩⟨7-битное число⟩ | `\parshape`

⟨специальное целое⟩ ⟶ `\spacefactor` | `\prevgraf`
| `\deadcycles` | `\insertpenalties`

⟨имя кода⟩ ⟶ `\catcode` | `\mathcode`
| `\lccode` | `\uccode` | `\sfcode` | `\delcode`

⟨шрифт⟩ ⟶ ⟨fontdef-элемент⟩ | `\font` | ⟨название члена⟩

⟨название члена⟩ ⟶ ⟨разряд шрифта⟩⟨4-бит.число⟩ ⟨разряд шрифта⟩ ⟶ `\textfont` | `\scriptfont` | `\scriptscriptfont`

⟨внутренний размер⟩ ⟶ ⟨параметр размер⟩ | ⟨специальный размер⟩
| `\lastkern` | ⟨dimendef-элемент⟩
| `\dimen`⟨8-битное число⟩
| `\fontdimen`⟨число⟩⟨шрифт⟩
| ⟨размер бокса⟩⟨8-битное число⟩

⟨специальный размер⟩ ⟶ `\prevdepth` | `\pagegoal` | `\pagetotal`
| `\pagestretch` | `\pagefilstretch` | `\pagefillstretch`
| `\pagefilllstretch` | `\pageshrink` | `\pagedepth`

⟨размер бокса⟩ ⟶ `\ht` | `\wd` | `\dp`

⟨внутренний клей⟩ ⟶ ⟨параметр клей⟩ | `\lastskip`
| ⟨skipdef-элемент⟩ | `\skip`⟨8-битное число⟩

⟨внутренний mu-клей⟩ ⟶ ⟨параметр mu-клей⟩ | `\lastskip`
| ⟨muskipdef-элемент⟩ | `\muskip`⟨8-битное число⟩

⟨countdef-элемент⟩ — это элемент команды, в которой текущее значение команды определено при помощи `\countdef`. Другие величины, ⟨dimendef-элемент⟩, ⟨skipdef-элемент⟩, ⟨muskipdef-элемент⟩, ⟨chardef-элемент⟩, ⟨mathchardef-элемент⟩, и ⟨toksdef -элемент⟩ , определены аналогично. ⟨fontdef -элемент⟩ ссылается на определение `\font`, или может быть переопределен идентификатором шрифта, называемым `\nullfont`. Когда ⟨countdef-элемент⟩ используется как целое число, он обозначает значение соответствующего `\count`-регистра. Аналогичные утверждения справедливы для ⟨dimendef-элемент⟩, ⟨skipdef-элемент⟩, ⟨muskipdef-элемент⟩. Когда ⟨chardef-элемент⟩ или ⟨mathchardef-элемент⟩ используются как целое число, они обозначают значения самих `\chardef` или `\mathchardef`. ⟨8-битное число⟩ — это ⟨число⟩, значение которого между 0 и $2^8 - 1 = 255$; ⟨7-битное число⟩ — это ⟨число⟩, значение которого между 0 и $2^7 - 1 = 127$, и так далее.

TEX позволяет, чтобы `\spacefactor` был внутренним целым числом только в горизонтальных модах, `\prevdepth` может быть внутренним размером только в вертикальных модах, `\lastskip` — это ⟨внутренний mu-клей⟩ только в математической моде, когда текущий математический список оканчивается mu-клеем, а `\lastskip` не может в таком случае быть внутренним клеем. Когда `\parshape` используется как внутреннее целое, оно обозначает только число управляемых

строк, а не их размеры или отступы. Когда текущая страница не содержит боксов, семь специальных размеров `\pagetotal`, `\pagestretch`, и т.п., все равны нулю, а `\pagegoal` в таких случаях равно `\maxdimen` (см. главу 15).

Из только что приведенных синтаксических правил можно вывести, что точно происходит с пробелами, если они соседствуют с числовыми величинами: TEX позволяет, чтобы перед ⟨число⟩ или ⟨размер⟩ находилось произвольно много пробелов и чтобы за ними следовало не больше одного пробела. Однако после ⟨число⟩ или ⟨размер⟩, которые заканчиваются нераскрываемой командой, нет возможного пробела. Например, если TEX встречает `\space\space24\space\space`, когда ищет ⟨число⟩, он заглатывает первые три пробела, но четвертый остается в живых. Аналогично, будет оставаться только по одному пробелу, когда `24pt\space\space`, `\pagegoal\space` и `\dimen24\space\space` рассматриваются как значения ⟨размер⟩.

Упражнение 24.2
Представляет ли `24\space\space pt` законный ⟨размер⟩?

Упражнение 24.3
Есть ли какая-нибудь разница между `+\baselineskip`, `1\baselineskip` и `- -\baselineskip`, когда TEX читает их как ⟨клей⟩?

Упражнение 24.4
Какой ⟨клей⟩ получается из `"DD DDPLUS2,5 \spacefactor\space`, предполагая соглашения начального TEX'а, когда `\spacefactor` равен 1000?

Давайте обратимся теперь к параметрам TEX'а, которые постепенно вводили предыдущие главы. Соберем их все вместе. ⟨Целый параметр⟩ — это один из следующих элементов:

`\pretolerance` (допуск плохости перед переносом)
`\tolerance` (допуск плохости после переноса)
`\hbadness` (плохость, свыше которой показан плохой h-бокс)
`\vbadness` (плохость, свыше которой показан плохой v-бокс)
`\linepenalty` (величина, добавляемая к плохости строкой абзаца)
`\hyphenpenalty` (штраф за разрыв строки после дискретного дефиса)
`\exhyphenpenalty` (штраф за разрыв строки после явного дефиса)
`\binoppenalty` (штраф за разрыв строки после бинарной операции)
`\relpenalty` (штраф за разрыв после математического отношения)
`\clubpenalty` (штраф за создание одинокой строки внизу страницы)
`\widowpenalty` (штраф за создание одинокой строки вверху страницы)
`\displaywidowpenalty` (то же самое, перед выделением)
`\brokenpenalty` (штраф за разрыв страницы после переносимой строки)
`\predisplaypenalty` (штраф за разрыв страницы прямо перед выделением)
`\postdisplaypenalty` (штраф за разрыв страницы сразу после выделения)
`\interlinepenalty` (штраф за разрыв страницы между строками)
`\floatingpenalty` (штраф за вставки, которые расщепляются)
`\outputpenalty` (штраф на текущий разрыв страницы)
`\doublehyphendemerits` (дефекты для последовательно разбитых строк)
`\finalhyphendemerits` (дефекты для предпоследней разбитой строки)
`\adjdemerits` (дефекты для соседствующих несовместимых строк)

`\looseness` (изменение числа строк в абзаце)
`\pausing` (>0 при паузе после каждой прочитанной строки)
`\tracingonline` (>0 при диагностике на терминале)
`\tracingmacros` (>0, если показывается раскрытие макрокоманд)
`\tracingstats` (>0, если показывается статистика памяти)
`\tracingparagraphs` (>0, если показывается разбиение строк)
`\tracingpages` (>0, если показывается разбиение страниц)
`\tracingoutput` (>0, если показываются выводимые боксы)
`\tracinglostchars` (>0, если показываются символы не из шрифта)
`\tracingcommands` (>0, если показываются команды)
`\tracingrestores` (>0, при показе деприсвоений по концу группы)
`\uchyph` (>0, если переносится слово с заглавный буквы)
`\globaldefs` (ненулевое, если подавляются `\global`-спецификации)
`\defaulthyphenchar` (значение `\hyphenchar`, когда шрифт загружается)
`\defaultskewchar` (`\skewchar`-значение, когда шрифт загружается)
`\escapechar` (сигнальный символ в выводе элементов команд)
`\endlinechar` (символ, помещаемый справа входной строки)
`\newlinechar` (символ, который начинает новую строку вывода)
`\maxdeadcycles` (верхняя граница `\deadcycles`)
`\hangafter` (изменения подвешенного отступа после множества строк)
`\fam` (текущий номер семейства)
`\mag` (масштаб увеличения, умноженный на 1000)
`\delimiterfactor` (пропорция переменных ограничителей $\times 1000$)
`\time` (текущее время дня в минутах после полуночи)
`\day` (текущий день месяца)
`\month` (текущий месяц года)
`\year` (текущий год от Рождества Христова)
`\showboxbreadth` (элементы уровня, когда показываются боксы)
`\showboxdepth` (максимальный уровень, когда показываются боксы)

Несколько первых из этих параметров выражены в единицах "плохости" и "штрафов", которые влияют на разбиение строк и страниц. Затем следуют дефекто-ориентированные параметры, которые по существу заданы в единицах "квадрата плохости", так что эти параметры имеют тенденцию к большим значениям. Наоборот, следующие несколько параметров (`\looseness`, `\pausing`, и т.д.) обычно имеют совсем маленькие значения (либо -1 или 0, либо 1 или 2). Список завершают разнообразные параметры. TEX вычисляет дату и время, когда начинает работу, если операционная система обеспечивает такую информацию, но после этого часы не тикают. Пользователь может изменить `\time` точно так же, как любой обычный параметр. Глава 10 отмечает, что `\mag` не должен изменяться после того, как TEX'у задано определенное увеличение.

⟨Параметр размера⟩ — это одно из следующего:

`\hfuzz` (максимум наполнения до сообщения о переполнении h-бокса)
`\vfuzz` (максимум наполнения до сообщения о переполнении v-бокса)
`\overfullrule` (ширина линейки переполнения бокса)
`\hsize` (ширина строки в горизонтальной моде)
`\vsize` (высота страницы в вертикальной моде)

`\maxdepth` (максимум глубины боксов на основных страницах)
`\splitmaxdepth` (максимум глубины боксов на разбитых страницах)
`\boxmaxdepth` (максимум глубины боксов на явных страницах)
`\lineskiplimit` (порог, где `\baselineskip` меняется на `\lineskip`)
`\delimitershortfall` (максимум места, непокрытого ограничителем)
`\nulldelimiterspace` (ширина нулевого ограничителя)
`\scriptspace` (дополнительный пробел после индекса)
`\mathsurround` (кернирование до и после математики в тексте)
`\predisplaysize` (длина текста, предшествующего выделению)
`\displaywidth` (длина строки для выделенного уравнения)
`\displayindent` (отступ строки для выделенного уравнения)
`\parindent` (ширина `\indent`)
`\hangindent` (величина подвешенного отступа)
`\hoffset` (горизонтальный сдвиг в `\shipout`)
`\voffset` (вертикальный сдвиг в `\shipout`)

Возможности для ⟨параметр клея⟩ следующие:

`\baselineskip` (желаемый клей между базовыми линиями)
`\lineskip` (внутренний клей, если `\baselineskip` невозможен)
`\parskip` (дополнительный клей прямо над абзацами)
`\abovedisplayskip` (дополнительный клей над выделениями)
`\abovedisplayshortskip` (то же, следующее за короткой строкой)
`\belowdisplayskip` (дополнительный клей под выделением)
`\belowdisplayshortskip` (то же, но за короткими строками)
`\leftskip` (клей слева выровненных строк)
`\rightskip` (клей справа выровненных строк)
`\topskip` (клей вверху основных страниц)
`\splittopskip` (клей вверху расщепленных страниц)
`\tabskip` (клей между выровненными элементами)
`\spaceskip` (клей между словами, если не нуль)
`\xspaceskip` (клей между предложениями, если не нуль)
`\parfillskip` (дополнительный `\rightskip` в конце абзаца)

Наконец, три допустимых элемента типа ⟨параметр mu-клея⟩:

`\thinmuskip` (тонкий пробел в математических формулах)
`\medmuskip` (средний пробел в математических формулах)
`\thickmuskip` (толстый пробел в математических формулах)

Все эти величины объяснены более подробно где-нибудь в этой книге, и можно использовать приложение I, чтобы найти, где.

TEX также имеет параметры, которые являются списками элементов. Такие параметры не входят в определение числа и тому подобных вещей, но мы можем также перечислить их сейчас, чтобы наша таблица параметров была полной. Каждый из приведенного ниже — это ⟨элементный параметр⟩:

`\output` (программа вывода пользователя)
`\everypar` (вставка в начале абзаца)
`\everymath` (вставка в начале математики в тексте)

`\everydisplay` (вставка в начале выделения математики)
`\everyhbox` (вставка в начале h-бокса)
`\everyvbox` (вставка в начале v-бокса)
`\everyjob` (вставка в начале работы)
`\everycr` (вставка после `\cr` или неизменного `\crcr`)
`\errhelp` (элементы, которые дополняют `\errmessage`)

Это дает всего 97 параметров всех пяти видов.

Упражнение 24.5
Объясните, как `\everyjob` может быть ненулевым, когда начинается работа.

Настало время вернуться к нашей начальной цели, а именно к изучению команд, которые подчиняются пищеварительным органам TEX'а. Многие команды выполняются одним и тем же способом безотносительно к текущей моде. Наиболее важные команды этого типа называются *присваиваниями*, поскольку присваивают новые величины значению команд или внутренним величинам TEX'а. Например, `\def\a{a}`, `\parshape=1 5pt 100pt`, `\advance\count20 by-1` и `\font\ff = cmff at 20pt` все являются присваиваниями и действуют одинаково во всех модах. Команды присваивания часто включают знак `=`, но не всегда этот знак обязателен. Вы можете пропустить его, если не возражаете против того, чтобы результирующий код TEX'а не выглядел, как присваивание.

⟨присваивание⟩ ⟶ ⟨немакро-присваивание⟩ | ⟨макро-присваивание⟩
⟨немакро-присваивание⟩ ⟶ ⟨простое присваивание⟩
| `\global`⟨немакро-присваивание⟩
⟨макро-присваивание⟩ ⟶ ⟨определение⟩ | ⟨префикс⟩⟨макро-присваивание⟩
⟨префикс⟩ ⟶ `\global` | `\long` | `\outer`
⟨равно⟩ ⟶ ⟨возможные пробелы⟩ | ⟨возможные пробелы⟩ `=`$_{12}$

Этот синтаксис показывает, что перед каждым присваиванием может быть добавлено `\global`, но только перед макроопределенными присваиваниями разрешено добавлять `\long` или `\outer`. Между прочим, если параметр `\globaldefs` является во время присваивания положительным, префикс `\global` прилагается автоматически. Но если во время присваивания `\globaldefs` отрицательно, то префикс `\global` игнорируется. Если `\globaldefs` равно нулю (как это обычно и есть), появление или непоявление префикса `\global` определяет то, является ли глобальным сделанное присваивание.

⟨определение⟩ ⟶ ⟨определить⟩⟨команда⟩⟨текст определения⟩
⟨определить⟩ ⟶ `\def` | `\gdef` | `\edef` | `\xdef`
⟨текст определения⟩ ⟶ ⟨параметрический текст⟩
⟨левая фигурная скобка⟩⟨сопоставленный текст⟩
⟨правая фигурная скобка⟩

Здесь ⟨команда⟩ обозначает элемент, который является либо командой, либо активным символом, ⟨левая фигурная скобка⟩ и ⟨правая фигурная скобка⟩ — это явные символьные элементы, номера категорий которых, соответственно, типов 1 и 2. ⟨Параметрический текст⟩ не содержит ни элемента левой фигурной скобки, ни элемента правой фигурной скобки и подчиняется правилам главы 20. Все

левые и правые фигурные скобки внутри сопоставленного текста должны быть правильно вложены. Команда `\gdef` эквивалентна `\global\def`, а `\xdef` эквивалентна `\global\edef`. TEX читает команду, параметрический текст и открывающую левую скобку, не раскрывая их. Он будет раскрывать сопоставленный текст и правую скобку только в случае `\edef` и `\xdef`.

Несколько команд, которые мы будем изучать ниже, имеют синтаксис, в чем то похожий на синтаксис определения; ⟨параметрический текст⟩ замещается произвольной последовательностью пробелов и командами `\relax`, а элемент ⟨левая фигурная скобка⟩ может быть неявным:

⟨заполнитель⟩ ⟶ ⟨возможные пробелы⟩
| ⟨заполнитель⟩`\relax`⟨возможные пробелы⟩
⟨общий текст⟩ ⟶ ⟨заполнитель⟩`{`⟨сопоставленный текст⟩
⟨правая фигурная скобка⟩

Основное назначение ⟨общий текст⟩ — задать сопоставленный текст внутри.

Имеется множество видов присваиваний, но они распадаются на сравнительно немного моделей, как указывается следующими синтаксическими правилами:

⟨простое присваивание⟩ ⟶ ⟨переменное присваивание⟩ | ⟨арифметика⟩
| ⟨код присваивания⟩ | ⟨let присваивание⟩
| ⟨краткое определение⟩ | ⟨fontdef-элемент⟩
| ⟨присваивание семейства⟩ | ⟨присваивание формы⟩
| `\read`⟨число⟩ `to` ⟨возможные пробелы⟩⟨команда⟩
| `\setbox`⟨8-битное число⟩⟨равно⟩⟨заполнитель⟩⟨бокс⟩
| `\font`⟨команда⟩⟨равно⟩⟨имя файла⟩⟨at-предложение⟩
| ⟨глобальное присваивание⟩
⟨переменное присваивание⟩ ⟶ ⟨целая переменная⟩⟨равно⟩⟨число⟩
| ⟨переменная размерности⟩⟨равно⟩⟨размер⟩
| ⟨переменная клея⟩⟨равно⟩⟨клей⟩
| ⟨переменная mu-клея⟩⟨равно⟩⟨mu-клей⟩
| ⟨переменная элемента⟩⟨равно⟩⟨общий текст⟩
| ⟨переменная элемента⟩⟨равно⟩⟨заполнитель⟩⟨переменная элемента⟩
⟨арифметика⟩ ⟶ `\advance`⟨целая переменная⟩⟨возможный `by`⟩⟨число⟩
| `\advance`⟨переменная размера⟩⟨возможный `by`⟩⟨размер⟩
| `\advance`⟨переменная клея⟩⟨возможный `by`⟩⟨клей⟩
| `\advance`⟨переменная mu-клея⟩⟨возможный `by`⟩⟨mu-клей⟩
| `\multiply`⟨числовая переменная⟩⟨возможный `by`⟩⟨число⟩
| `\divide`⟨числовая переменная⟩⟨возможный `by`⟩⟨число⟩
⟨возможный `by`⟩ ⟶ `by` | ⟨возможные пробелы⟩
⟨целая переменная⟩ ⟶ ⟨целый параметр⟩
| ⟨count-определенный элемент⟩
| `\count`⟨8-битное число⟩
⟨переменная размера⟩ ⟶ ⟨параметр размера⟩
| ⟨dimen-определенный элемент⟩
| `\dimen`⟨8-битное число⟩
⟨переменная клея⟩ ⟶ ⟨параметр клея⟩
| ⟨skip-определенный элемент⟩

| `\skip`⟨8-битное число⟩
⟨переменная mu-клея⟩ ⟶ ⟨параметр mu-клея⟩
| ⟨muskip-определенный элемент⟩
| `\muskip`⟨8-битное число⟩
⟨элементная переменная⟩ ⟶ ⟨параметр элемента⟩
| ⟨toks-определенный элемент⟩
| `\toks`⟨8-битное число⟩
⟨числовая переменная⟩ ⟶ ⟨целая переменная⟩
| ⟨переменная размера⟩
| ⟨переменная клея⟩ | ⟨переменная mu-клея⟩
⟨присваивание кода⟩ ⟶ ⟨имя кода⟩⟨7-битное число⟩⟨равно⟩⟨число⟩
⟨let присваивание⟩ ⟶ `\futurelet`⟨команда⟩⟨элемент⟩⟨элемент⟩
| `\let`⟨команда⟩⟨равно⟩⟨один возможный пробел⟩⟨элемент⟩
⟨краткое определение⟩ ⟶ `\chardef`⟨команда⟩⟨равно⟩⟨8-битное число⟩
| `\mathchardef`⟨команда⟩⟨равно⟩⟨15-битное число⟩
| ⟨определение регистра⟩⟨команда⟩⟨равно⟩⟨8-битное число⟩
⟨определение регистра⟩ ⟶ `\countdef` | `\dimendef` | `\skipdef`
| `\muskipdef` | `\toksdef`
⟨присваивание семейства⟩ ⟶ ⟨член семейства⟩⟨равно⟩⟨шрифт⟩
⟨присваивание формы⟩ ⟶ `\parshape`⟨равно⟩⟨число⟩⟨размер формы⟩

⟨Число⟩ в конце присваивания кода не должно быть отрицательным, за исключением случая, когда присваивается `\delcode`. Более того, это ⟨число⟩ должно быть не более 15 для `\catcode`, 32768 для `\mathcode`, 127 для `\lccode` `\uccode`, 32767 для `\sfcode`, и $2^{24}-1$ для `\delcode`. В присваивании формы, для которого ⟨число⟩ — это n, ⟨размеры формы⟩ являются пустыми, если $n \leq 0$, иначе они состоят из $2n$ последовательных размеров. TEX не раскрывает элементы, когда просматривает аргументы команд `\let` и `\futurelet`.

▸ **Упражнение 24.6**
Ранее в этой главе мы обсуждали различие между явными и неявными символьными элементами. Объясните, как можно превратить команду `\cs` в неявный пробел, используя (a) `\futurelet`, (b) `\let`.

Все упомянутые присваивания подчиняются групповой структуре TEX'a, то есть измененные величины будут восстановлены в своих прежних значениях, когда окончится текущая группа, если только изменения не были глобальными. Следующие присваивания отличаются от предыдущих, поскольку они глобально действуют на таблицы шрифтов TEX'a и на таблицы переноса, или они действуют на переменные такой природы, что группирование было бы неуместным. Во всех следующих случаях присутствие или отсутствие префикса `\global` не имеет влияния.

⟨глобальное присваивание⟩ ⟶ ⟨присваивание шрифта⟩
| ⟨присваивание переноса⟩
| ⟨присваивание размера бокса⟩
| ⟨присваивание режима взаимодействия⟩
| ⟨частное присваивание⟩

⟨присваивание шрифта⟩ ⟶ `\fontdimen`⟨число⟩⟨шрифт⟩⟨равно⟩⟨размер⟩
| `\hyphenchar`⟨шрифт⟩⟨равно⟩⟨число⟩
| `\skewchar`⟨шрифт⟩⟨равно⟩⟨число⟩
⟨at-предложение⟩ ⟶ `at` ⟨размер⟩ | `scaled` ⟨число⟩ | ⟨возможный пробел⟩
⟨присваивание переноса⟩ ⟶ `\hyphenation`⟨общий текст⟩
| `\patterns`⟨общий текст⟩
⟨присваивание размера бокса⟩ ⟶ ⟨размер бокса⟩⟨8-битное число⟩
⟨равно⟩⟨размер⟩
⟨присваивание режима взаимодействия⟩ ⟶ `\errorstopmode`
| `\scrollmode` | `\nonstopmode`
| `\batchmode`
⟨частное присваивание⟩ ⟶ ⟨специальное целое число⟩⟨равно⟩⟨число⟩
| ⟨специальный размер⟩⟨равно⟩⟨размер⟩

Когда присваивается значение `\fontdimen`, ⟨число⟩ должно быть положительным и не больше числа параметров в метрическом файле шрифта, если только эта информация шрифта не только что загружена в память TEX'а. В последнем случае вам разрешено увеличить число параметров (см. приложение F). ⟨Специальное целое⟩ и ⟨специальный размер⟩ были описаны выше, когда мы обсуждали внутренние целые числа и размеры. Когда `\prevgraf` устанавливается в ⟨число⟩, число не должно быть отрицательным.

В TEX'е нет стандарта на синтаксис для ⟨имя файла⟩, поскольку различные операционные системы придерживаются различных соглашений. Вы должны узнать у местных системщиков, как они решили реализовать имена файлов. Однако, следует придерживаться универсальных принципов. ⟨Имя файла⟩ должно состоять из ⟨возможные пробелы⟩, за которыми следуют явные символьные элементы (после раскрытия). Именем файла должна быть последовательность шести или менее обычных букв и/или цифр, что работает по существу одинаково во всех реализациях TEX'а. В именах файлов заглавные буквы не считаются эквивалентными их строчным парам. Например, если вы ссылаетесь на шрифты `cmr10` и `CMR10`, TEX не заметит какого-нибудь сходства между ними, хотя он мог ввести один и тот же метрический файл для обоих шрифтов.

TEX специально заботится, чтобы конструкции типа `\chardef\cs=10\cs` и `\font\cs=name\cs` не раскрывали второй `\cs` до тех пор, пока не сделано присваивание.

Наше обсуждение присваиваний завершено, за тем исключением, что присваивание `\setbox` включает величину ⟨бокс⟩, которая еще не определена. Приведем ее синтаксис:

⟨бокс⟩ ⟶ `\box`⟨8-битное число⟩ | `\copy`⟨8-битное число⟩
| `\lastbox` | `\vsplit`⟨8-битное число⟩ `to` ⟨размер⟩
| `\hbox`⟨спецификация бокса⟩`{`⟨горизонтальный материал⟩`}`
| `\vbox`⟨спецификация бокса⟩`{`⟨вертикальный материал⟩`}`
| `\vtop`⟨спецификация бокса⟩`{`⟨вертикальный материал⟩`}`
⟨спецификация бокса⟩ ⟶ `to` ⟨размер⟩⟨заполнитель⟩
| `spread` ⟨размер⟩⟨заполнитель⟩ | ⟨заполнитель⟩

Операция `\lastbox` не допускается в математических модах, а также и в вертикальной моде, когда основной вертикальный список целиком занесен на текущую

страницу. Но она разрешена в горизонтальных модах и внутренней вертикальной моде. В таких модах она ссылается на последний элемент текущего списка (и перемещает его) в том случае, если последний элемент является h-боксом или v-боксом.

Три последние альтернативы для ⟨бокс⟩ знакомят нас с новой ситуацией: ⟨горизонтальный материал⟩ в `\hbox` и ⟨вертикальный материал⟩ в `\vbox` не могут просто поглощаться одной командой как ⟨8-битное число⟩ или ⟨размер⟩. Может быть придется выполнить тысячи команд, прежде чем этот бокс сконструируется и прежде чем может быть завершена команда `\setbox`.

Покажем, что же реально происходит. Команда типа

`\setbox`⟨число⟩`=\hbox to`⟨размер⟩`{`⟨горизонтальный материал⟩`}`

указывает TEX'у выразить в числах ⟨число⟩ и ⟨размер⟩ и поместить эти значения в "стек" для сохранения. Затем TEX читает "`{`" (которая, как объяснялось ранее, обозначает явный или неявный символ начала группы), и это начинает новый уровень группирования. В этой точке TEX входит в ограниченную горизонтальную моду и переходит к выполнению команд в этой моде. Может быть сконструирован сколь угодно сложный бокс. Тот факт, что этот бокс по существу предназначен для команды `\setbox`, не влияет на поведение TEX'а при построении этого бокса. В конце концов, когда появляется закрывающая "`}`", TEX восстанавливает те же значения, которые были изменены присваиваниями в только что оконченной группе. Затем он упаковывает h-бокс (используя размеры, которые хранились в стеке) и завершает команду `\setbox`, возвращаясь в моду, в которой он был к началу `\setbox`.

Давайте теперь рассмотрим другие команды, которые выполняются в основном одинаково, независимо от текущей моды TEX'а.

- `\relax`. Это легко: TEX ничего не делает.

- `}`. Это немного труднее, поскольку зависит от текущей группы. TEX должен работать в группе, которая началась с `{` и знает, почему он начал эту группу. Так что он производит соответствующие окончанию действия, разрушая последствия неглобальных присваиваний, и покидает группу. В этой точке TEX может покинуть текущую моду и вернуться в предыдущую.

- `\begingroup`. Когда TEX видит эту команду, он входит в группу, которая должна окончиться `\endgroup`, а не `}`. Мода не изменяется.

- `\endgroup`. TEX в настоящий момент должен обрабатывать группу, которая началась с `\begingroup`. Величины, которые были в этой группе изменены неглобальными присваиваниями, восстанавливают свои прежние значения. TEX покидает группу, но остается в той же моде.

- `\showlists`, `\showbox`⟨8-битное число⟩, `\showthe`⟨внутренняя величина⟩ `\show`⟨элемент⟩. Эти команды призваны помочь вам разобраться в том, что TEX думает о том, что он делает. Элементы, следующие за `\showthe`, должны быть чем либо, что может следовать за `\the`, как объяснялось в главе 20.

Упражнение 24.7

Посмотрите в главе 20 описание того, что может следовать за `\the`, и

сконструируйте формальный синтаксис, который определяет внутреннюю величину тем способом, который соответствует синтаксическим правилам, которые мы обсудили.

■ `\shipout`⟨бокс⟩. После того, как ⟨бокс⟩ сформирован — возможно, конструируя его явно и изменяя моды, как ранее объяснялось для `\hbox` — его содержание посылается в `dvi-` файл (см. главу 23).

■ `\ignorespaces`⟨возможные пробелы⟩. TEX читает (и раскрывает) элементы, ничего не делая, пока не встретит то, что не является элементом пробела.

■ `\afterassignment`⟨элемент⟩. ⟨Элемент⟩ хранится в специальном месте и будет вставлен во входные данные сразу после того, как будет исполнена следующая команда присваивания. Присваивание не должно следовать немедленно. Если перед следующим присваиванием выполняется другое `\afterassignment`, то второе перекроет первое. Если следующим присваиванием будет `\setbox` и если присваиваемый ⟨бокс⟩ — это `\hbox`, `\vbox` или `\vtop`, то ⟨элемент⟩ будет вставляться сразу после `{` , а не после `}`. Он вставится также перед любыми элементами, которые вставляются командами `\everyhbox` или `\everyvbox`.

■ `\aftergroup`⟨элемент⟩. ⟨Элемент⟩ хранится в стеке TEX'а. Он будет вставляться во входную информацию сразу после того, как закончится текущая группа и будут разрушены ее локальные присваивания. Если в одной и той же группе встречаются несколько команд `\aftergroup`, соответствующие команды будут просматриваться в том же порядке. Например, `{\aftergroup\a\aftergroup\b}` дает `\a\b`.

■ `\uppercase`⟨общий текст⟩, `\lowercase`⟨общий текст⟩. В этих командах сопоставленный текст в общем тексте преобразуется в заглавную или строчную форму, используя таблицу `\uccode` или `\lccode`, как объяснялось в главе 7. Никакого раскрытия не делается. Затем TEX читает этот сопоставленный текст снова.

■ `\message`⟨общий текст⟩, `\errmessage`⟨общий текст⟩. В этих командах сопоставленный текст (с раскрытием) пишется на терминал пользователя, используя формат сообщений об ошибках в случае `\errmessage`. В последнем случае элементы `\errhelp` будут показаны, если они не пустые и если пользователь просит о помощи.

■ `\openin`⟨4-бит.число⟩⟨равно⟩⟨имя файла⟩, `\closein`⟨4-бит.число⟩. Выходной поток открыт или закрыт для команд `\read`, как объяснялось в главе 20.

■ `\immediate\openout`⟨4-бит.число⟩⟨равно⟩⟨имя файла⟩,а также `\immediate \closeout`⟨4-бит.число⟩. Указанный выходной поток открыт или закрыт для команд `\write`, как объяснялось в главе 21.

■ `\immediate\write`⟨число⟩⟨общий текст⟩. Сопоставленный текст пишется в файл, который соответствует указанному номеру потока, при условии, что такой файл открыт. В противном случае он пишется на терминал пользователя и в протокольный файл. (См. главу 21; вывод на терминал опускается, если ⟨число⟩ отрицательное).

Это завершает список независимых от моды команд, то есть команд, на которые прямо не действует список, который строит TEX. Когда TEX находится в вертикальной или внутренней вертикальной моде, он строит вертикальный список. Когда TEX находится в горизонтальной или частной горизонтальной моде,

он строит горизонтальный список. Когда TEX находится в математической или выделенной математической моде, он строит — угадайте, что? — математический список. В каждом из этих случаев мы можем говорить "текущий список", и существует несколько команд, которые выполняются, по существу, одним и тем же способом независимо от моды, за исключением того, что имеют дело с различными видами списков:

- `\openout`⟨4-бит.число⟩⟨равно⟩⟨имя файла⟩, `\closeout`⟨4-бит.число⟩, а также `\write`⟨число⟩⟨общий текст⟩. Эти команды записываются в "whatsit", который добавляется к текущему списку. Команды будут выполняться позднее во время какого-либо `\shipout`, который применяется к этому списку, если только этот список не является частью бокса внутри проводников.

- `\special`⟨общий текст⟩. Сопоставленный текст раскрывается и помещается в "whatsit", который добавляется к текущему списку. Текст в конце концов появится в `dvi`-файле как инструкция для последующего матобеспечения (см. главу 21).

- `\penalty`⟨число⟩. Штраф, равный указанному числу, добавляется к текущему списку. В вертикальной моде TEX также выполняет обязанности планировщика страниц (см. ниже).

- `\kern`⟨размер⟩, `\mkern`⟨mu-размер⟩. Керн заданного размера добавляется к текущему списку. В вертикальной моде он обозначает вертикальный пробел, иначе он обозначает горизонтальный пробел. `\mkern` разрешен только в математических модах.

- `\unpenalty`, `\unkern`, `\unskip`. Последний элемент текущего списка должен иметь, соответственно, тип штрафа, керна или клея (возможно, включая проводники), и этот элемент убирается из списка. Однако, как и `\lastbox`, эти команды не допускаются в вертикальной моде, если существовавший до сих пор вертикальный список был целиком занесен на текущую страницу, поскольку TEX никогда не убирает элементы с текущей страницы.

- `\mark`⟨общий текст⟩. Сопоставленный текст раскрывается и помещается в метку, которая добавляется к текущему списку. Текст в конце концов может стать текстом замены для `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark` и/или `\splitbotmark`, если эта метка оказалась в вертикальном списке. (Метки могут появиться в горизонтальных и математических списках, но там они не действуют, если только они не "переселяются" из своего списка. Процесс переселения обсуждается ниже и в главе 25.)

- `\insert`⟨8-битное число⟩⟨заполнитель⟩`{`⟨вертикальный материал⟩`}`; Это ⟨8-битное число⟩ не должно быть 255. "`{`" указывает TEX'у войти во внутреннюю вертикальную моду и на новый уровень группирования. Когда встречается парная "`}`", в вертикальный список помещается вставка, которая добавляется к текущему списку, используя те значения `\splittopskip`, `\splitmaxdepth`, и `\floatingpenalty`, которые были текущими в только что законченной группе. (См. главу 15.) Эта вставка приводит к вставке на странице, только если она появляется в основном вертикальном списке TEX'а, так что вставка будет вынуждена "переселиться" туда, если она начиналась в горизонтальном или математическом списке. TEX также выполняет функции планировщика страницы (см. ниже) после того, как `\insert` добавлена в вертикальной моде.

■ `\vadjust`⟨заполнитель⟩`{`⟨вертикальный материал⟩`}`. Эта команда аналогична команде `\insert`. Созданный вертикальный список помещается в регулировку, которая добавляется к текущему списку. Однако, `\vadjust` не разрешено в вертикальных модах. Когда регулировка переселяется из горизонтального списка в вертикальный, то вертикальный список внутри регулировки "развертывается" и помещается непосредственно в охватывающий список.

* * *

Почти все, что мы до сих пор в этой главе обсудили, могло с одинаковым успехом появиться и в главе "Обзор горизонтальной моды", и в главе "Обзор математической моды", поскольку TEX трактует все рассмотренные до сих пор команды одним и тем же образом, независимо от текущей моды. Главы 25 и 26 будут значительно короче этой главы, так как нет необходимости повторять все независимые от моды правила.

Но сейчас мы подошли к командам, которые зависят от моды. Завершим эту главу обсуждением того, что TEX делает с остающимися командами, когда находится в вертикальной или внутренней вертикальной моде.

Характерной для вертикальной моды является операция построения страницы, описанная в главе 15. TEX периодически берет материал, который положен в основной вертикальный список и передвигает его из "списка дополнительного вклада" на "текущую страницу". В таких случаях может быть вызвана программа вывода. Мы будем говорить, что TEX *выполняет функции планировщика страницы*, всякий раз, когда он старается опустошить текущий список дополнительного вклада. Понятие списка дополнительного вклада существует только в самой внешней вертикальной моде, поэтому ничего не происходит, когда TEX выполняет функции планировщика страницы во внутренней вертикальной моде.

Также для вертикальной моды характерным является междустрочный клей, который вставляется перед боксами на основании `\prevdepth`, `\baselineskip`, `\lineskip` и `\lineskiplimit`, как объяснялось в главе 12. Если команда изменяет `\prevdepth`, то этот факт специально указывается ниже. Значение `\prevdepth` изначально устанавливается в -1000 pt, специальное значение, которое задерживает междустрочный клей каждый раз, когда TEX начинает формировать вертикальный список, за исключением случая `\halign` и `\noalign`, когда соглашения междустрочного клея для внешнего списка продолжаются во внутреннем списке.

■ `\vskip`⟨клей⟩, `\vfil`, `\vfill`, `\vss`, `\vfilneg`. Клей добавляется к текущему вертикальному списку.

■ ⟨проводники⟩⟨бокс или линейка⟩⟨вертикальный пропуск⟩. В этой команде величина ⟨вертикальный пропуск⟩ отсылает к одной из пяти только что упомянутых команд, добавляющих клей. Приведем формальный синтаксис для величин ⟨проводники⟩ и ⟨бокс или линейка⟩:

⟨проводники⟩ ⟶ `\leaders` | `\cleaders` | `\xleaders`
⟨бокс или линейка⟩ ⟶ ⟨бокс⟩ | ⟨вертикальная линейка⟩
| ⟨горизонтальная линейка⟩
⟨вертикальная линейка⟩ ⟶ `\vrule`⟨спецификация линейки⟩
⟨горизонтальная линейка⟩ ⟶ `\hrule`⟨спецификация линейки⟩
⟨спецификация линейки⟩ ⟶ ⟨возможные пробелы⟩
| ⟨размер линейки⟩⟨спецификация линейки⟩

⟨размер линейки⟩ ⟶ `width` ⟨размер⟩ | `height` ⟨размер⟩
| `depth` ⟨размер⟩

Клей, который производится проводниками, добавляется к текущему списку.

■ ⟨элемент пробела⟩. Пробелы в вертикальных модах не действуют.

■ ⟨бокс⟩. Создается бокс, и если результат пустой, то ничего не происходит. Иначе текущий вертикальный список получает (1) междустрочный клей, за которым следует (2) новый бокс, за которым следует (3) вертикальный материал, который переселился из нового бокса (если ⟨box⟩ был командой `\hbox`). Затем `\prevdepth` устанавливает глубину нового бокса, и TEX выполняет функции планировщика страницы.

■ `\moveleft`⟨размер⟩⟨бокс⟩, `\moveright`⟨размер⟩⟨бокс⟩. Это действует точно также, как обычная команда ⟨box⟩, но новый бокс, который добавляется к вертикальному списку, сдвигается налево или направо на заданную величину.

■ `\unvbox`⟨8-битное число⟩, `\unvcopy`⟨8-битное число⟩. Если указанный регистр бокса пустой, то ничего не происходит. В противном случае этот регистр должен содержать v-бокс. Вертикальный список внутри этого бокса добавляется к текущему вертикальному списку без всякого изменения. Значение `\prevdepth` не действует. Регистр бокса становится пустым после `\unvbox`, но остается неизменным после `\unvcopy`.

■ ⟨горизонтальная линейка⟩. Эта линейка добавляется к текущему списку. Затем `\prevdepth` устанавливается в -1000 pt. Это будет препятствовать междустрочному клею, когда к списку добавляется следующий бокс.

■ `\halign`⟨спецификация бокса⟩`{`⟨выравниваемый материал⟩`}`. Этот выравниваемый материал состоит из преамбулы, за которой следуют ноль или более строчек для выравнивания; см. главу 22. TEX входит на новый уровень группирования, представленный скобками `{` и `}`, внутри которого будут ограничены изменения `\tabskip`. Выравниваемый материал также содержит между строками возможные `\noalign`⟨заполнитель⟩`{`⟨вертикальный материал⟩`}`. Это добавляет уровень группирования. TEX действует во внутренней вертикальной моде, когда обрабатывает материал в группах `\noalign` и когда добавляет строки выравнивания. Результирующий внутренний вертикальный список будет добавлен к охватывающему вертикальному списку после того, как выравнивание закончится, и будет работать планировщик страницы. Значение `\prevdepth` во время `\halign` используется в начале внутреннего вертикального списка, а когда выравнивание окончено, конечное значение `\prevdepth` переносится в охватывающий вертикальный список, так что междустрочный клей вычисляется правильно и в начале, и в конце выравнивания. TEX также входит на дополнительный уровень группирования, когда обрабатывает каждый элемент выравнивания, и в это время он действует в частной горизонтальной моде. Конкретные элементы будут горизонтально боксированы как часть конечного выравнивания, и их вертикальный материал переселится в охватывающий вертикальный список. Команды `\noalign`, `\omit`, `\span`, `\cr`, `\crcr` и `&` (где `&` обозначает явный или неявный символ категории 4) перехватываются процессом выравнивания по дороге в желудок TEX'а, так что они не появляются в качестве команд в желудке, если только TEX не потерял след того, к какому выравниванию они принадлежат.

■ `\indent`. Клей `\parskip` добавляется к текущему списку, если TEX не находится в внутренней вертикальной моде и если текущий список не пуст. Затем TEX входит в общую горизонтальную моду, начиная горизонтальный список с пустого h-бокса, ширина которого равна `\parindent`. Во входной поток вставляются элементы `\everypar`. Работает планировщик страниц. Когда абзац завершен, горизонтальная мода окончится, как это описано в главе 25.

■ `\noindent`. Это точно также, как `\indent`, за тем исключением, что TEX приступает к горизонтальной моде с пустого списка, а не с отступа.

■ `\par`. Примитивная команда `\par` не действует, когда TEX в вертикальной моде, за исключением того, что планировщик страницы работает, когда что-нибудь присутствует в списке дополнительного вклада и параметры формы параграфа очищаются.

■ `{`. Символьный элемент категории 1 или команда типа `\bgroup`, которая `\let`-эквивалентна такому символьному элементу, указывает TeX'у начать новый уровень группирования.Когда такая группа заканчивается — знаком `}` — TEX разрушает неглобальные присваивания, не покидая моды, в которой в это время находится.

■ Некоторые команды несовместимы с вертикальной модой, поскольку они существенно горизонтальны. Когда следующие команды появляются в вертикальных модах, они указывают TEX'у начать новый абзац:

⟨горизонтальная команда⟩ ⟶ ⟨буква⟩ | ⟨другой символ⟩
| `\char` | ⟨chardef-знак⟩ | `\unhbox`
| `\unhcopy` | `\valign` | `\vrule` | `\hskip`
| `\hfil` | `\hfill` | `\hss` | `\hfilneg`
| `\accent` | `\discretionary` | `\-` | `\␣` | `$`

Здесь ⟨буква⟩ и ⟨другой символ⟩ обозначают явные или неявные элементы категорий 11 и 12. Если любой из таких символов встречается как команда в вертикальной или внутренней вертикальной моде, TEX автоматически выполняет команду `\indent`, как объяснялось выше. Это введет горизонтальную моду с элементами `\everypar` на входе, после чего TEX будет снова посмотрит горизонтальную команду.

■ `\end`. Эта команда не разрешена во внутренней вертикальной моде. В нормальной вертикальной моде это прерывает TEX, если основной вертикальный список пуст и `\deadcycles=0`. Иначе TEX возвращает команду `\end`, так что она может быть прочитана снова, а затем выполняет функции планировщика страницы после добавления комбинации бокс/клей/штраф, которая заставит действовать программу вывода. (См. конец главы 23.)

■ `\dump`. (Разрешена только в `INITEX`, а не в производственных версиях TEX'а.) Эта команда трактуется в точности как `\end`, но не должна появляться внутри группы. Она выводит форматный файл, который может быть загружен в память TEX'а со сравнительно большой скоростью, чтобы восстановить текущее состояние.

■ Нечто, что не указано выше. Если в вертикальной моде встречается любая другая примитивная команда TEX'а, будет дано сообщение об ошибке и TEX попытается восстановить разумные действия. Например, если появляется символ

верхнего или нижнего индекса или дана любая другая команда, свойственная математике, TEX будет пытаться вставить `$` (который начнет абзац и войдет в математическую моду). Но если в вертикальной моде появляется совсем перепутанный элемент типа `\endcsname`, `\omit`, `\eqno` или `#`, TEX после сообщения об ошибке будет просто игнорировать его. Можно поразвлекаться, печатая какие-нибудь глупости, чтобы посмотреть, что получится. (Сначала скажите `\tracingall`, как объяснялось в главе 27, чтобы получить максимальную информацию.)

Первая и самая бросающаяся в глаза черта —
это Вертикальность композиции,
как противоположность Горизонтальности
всех прежних структурных стилей.

The first and most striking feature
is the Verticality of composition,
as opposed to the Horizontality
of all anterior structural modes.

—COCKBURN MUIR, *Pagan or Christian?* (1860)

Иногда, закончив книгу,
я делаю ее полный обзор.

Sometimes when I have finished a book
I give a summary of the whole of it.

— ROBERT WILLIAM DALE, *Nine Lectures on Preaching*(1878)

25

Обзор горизонтальной моды

Продолжим обзор, начатый в главе 24. Давайте исследуем подробно, что может делать пищеварительный процесс TEX'а, когда TEX строит списки в горизонтальной или частной горизонтальной моде.

* * *

Три звездочки, точно такие, как те, которые появились здесь, можно найти неподалеку от конца главы 24. Все, что предшествует в той главе трем звездочкам, с тем же успехом применимо к горизонтальной моде, что и к вертикальной, так что нет необходимости повторять все эти правила. В частности, глава 24 разъясняет команды присваивания и рассказывает, как керны, штрафы, метки, вставки, регулировки и "whatsits" помещаются в горизонтальные списки. Теперь наша цель рассмотреть команды, которые имеют специфический горизонтальный аромат, в том смысле, что в горизонтальной моде ведут себя иначе, чем в вертикальной или математической.

Характерным для горизонтальной моды является "коэффициент пробела", который изменяет величину промежутков, как это описано в главе 12. Если команда изменяет величину коэффициента пробела `\spacefactor`, то этот факт здесь специально отмечается. Коэффициент пробела изначально, когда TEX начинает формировать горизонтальный список, устанавливается равным 1000, за исключением случая `\valign` и `\noalign`, когда коэффициент пробела внешнего списка продолжает действовать во внутреннем списке.

- `\hskip`⟨клей⟩, `\hfil`, `\hfill`, `\hss`, `\hfilneg`. Клей добавляется к текущему горизонтальному списку.

- ⟨проводники⟩⟨бокс или линейка⟩⟨горизонтальный пропуск⟩. В таких конструкциях ⟨горизонтальный пропуск⟩ — это одна из пяти добавляющих клей команд, только что упомянутых. Формальный синтаксис для ⟨проводники⟩ и для ⟨бокс или линейка⟩ дается в главе 24. Добавляется клей, который производят проводники.

- ⟨элемент пробела⟩. Пробелы добавляют клей к текущему списку. Точная величина клея зависит от величины `\spacefactor`, текущего шрифта и параметров `\spaceskip` и `\xspaceskip`, как описано в главе 12.

- \␣. Командный пробел добавляет клей к текущему списку, используя ту же самую величину, которую вставляет ⟨элемент пробела⟩, когда коэффициент пробела равен 1000.

- ⟨бокс⟩. Создается бокс и, если результат пустой, ничего не происходит. В противном случае новый бокс добавляется к текущему списку и коэффициент пробела устанавливается в 1000.

- `\raise`⟨размер⟩⟨бокс⟩, `\lower`⟨размер⟩⟨бокс⟩. Это действует так же, как обычный ⟨бокс⟩, но новый бокс, который добавляется к текущему списку, сдвигается вверх или вниз на указанную величину.

- `\unhbox`⟨8-битное число⟩, `\unhcopy`⟨8-битное число⟩. Если указанный регистр бокса пустой, то ничего не происходит. Если регистр содержит h-бокс, то горизонтальный список внутри этого бокса добавляется к текущему горизонтальному списку без каких-либо изменений. Значение `\spacefactor` не имеет влияния. Регистр бокса становится пустым после `\unhbox`, но остается неизменным после `\unhcopy`.

■ ⟨вертикальная линейка⟩. Данная линейка добавляется к текущему списку и `\spacefactor` устанавливается в 1000.

■ `\valign`⟨спецификация бокса⟩`{`⟨выравниваемый материал⟩`}`. Этот выравниваемый материал состоит из преамбулы, за которым следует ноль или более колонок выравнивания (см. главу 22). TEX входит на новый уровень группирования, представленный скобками `{` и `}`, внутри которого будут ограничены изменения `\tabskip`. В выравниваемом материале между колонками может также встретиться `\noalign`⟨заполнитель⟩`{`⟨материал горизонтальной моды⟩`}`, что добавляет еще один уровень группирования. Когда TEX обрабатывает материал групп `\noalign` и добавляет колонки выравнивания, он действует в частной горизонтальной моде. После того, как выравнивание завершено, полученный внутренний горизонтальный список будет добавлен к охватывающему горизонтальному списку. Значение `\spacefactor` во время `\valign` используется в начале внутреннего горизонтального списка и конечное значение `\spacefactor`, когда выравнивание оканчивается, переносится в охватывающий горизонтальный список. Коэффициент пробела устанавливается в 1000 после каждой колонки, следовательно, влияет на результат только в группах `\noalign`. TEX также входит на дополнительный уровень группирования, когда обрабатывает каждый конкретный элемент выравнивания, при этом он действует во внутренней вертикальной моде. Конкретные элементы будут помещены в v-бокс как часть окончательного выравнивания.

■ `\indent`. К текущему списку добавляется пустой бокс, ширина которого равна `\parindent` и коэффициент пробела устанавливается в 1000.

■ `\noindent`. Команда в горизонтальной моде не работает.

■ `\par`. Примитивная команда `\par`, в начальном TEX'е также называемая `\endgraf`, в частной горизонтальной моде не делает ничего. Но она заканчивает горизонтальную моду. Текущий список заканчивается, выполнив `\unskip \penalty10000 \hskip\parfillskip`, затем он разбивается на строки, как объяснялось в главе 14, и TEX возвращается к охватывающей вертикальной или внутренней вертикальной моде. Строки абзаца добавляются к охватывающему вертикальному списку, пересыпаемые междустрочным клеем и междустрочными штрафами и с переселением вертикального материала, который был в горизонтальном списке. Затем TEX выполняет функции планировщика страниц.

■ `{`. Символьный элемент категории 1, или команда типа `\bgroup`, которая `\let`-равна такому символьному элементу, указывает TEX'у начать новый уровень группирования. Когда такая группа кончается — с помощью "`}`" — TEX разрушает действия неглобальных присваиваний, не покидая моду, в которой находится в это время.

■ Некоторые команды несовместимы с горизонтальной модой, потому что они существенно вертикальны. Когда следующие команды появляются в общей горизонтальной моде, они указывают TEX'у завершить текущий абзац:

⟨вертикальная команда⟩ ⟶ `\unvbox` | `\unvcopy` | `\halign` | `\hrule`
| `\vskip` | `\vfil` | `\vfill` | `\vss` | `\vfilneg` | `\end` | `\dump`

Появление вертикальной команды в частной горизонтальной моде запрещено, но в нормальной горизонтальной моде она указывает TEX'у вставить во входные данные `par`. После прочтения и раскрытия этого элемента `par`, TEX будет снова

смотреть вертикальную команду. (TEX'ом будет использовано текущее значение команды `\par`; `par` может не означать TEXовский примитив `\par`.)

■ ⟨буква⟩, ⟨другой символ⟩, `\char`⟨8-битное число⟩, ⟨chardef элемент⟩. Это наиболее общий тип "команды" в горизонтальных модах. Она указывает TEX'у добавить символ к текущему списку, используя текущий шрифт. Если встречаются две или более команд такого типа подряд, TEX обрабатывает их все, как одну, преобразуя пары символов в лигатуры и/или вставляя керны, как указано в информации шрифта. Каждая символьная команда для символов с номерами от 0 до 127 регулирует `\spacefactor` с помощью таблицы `\sfcode`, как описывалось в главе 14. Символы с номерами от 128 до 255 устанавливают коэффициент пробела в 1000. В общей горизонтальной моде добавляется элемент `\discretionary{}{}{}` после символа, код которого является `\hyphenchar` его шрифта, или после лигатуры, образованной из последовательности, которая оканчивается таким символом.

■ `\accent`⟨8-битное число⟩⟨возможные присваивания⟩. В этой конструкции ⟨возможные присваивания⟩ обозначают ноль или несколько команд присваивания. Если за присваиваниями не следует ⟨символ⟩, где ⟨символ⟩ обозначает любую команду, которая только что обсуждалась в предыдущем абзаце, TEX трактует `\accent`, как если бы он был `\char`, за тем исключением, что коэффициент пробела устанавливается в 1000. Иначе символ, который следует за присваиванием, акцентируется символом, который соответствует 8-битному числу. (Цель вмешательства присваиваний — разрешить, чтобы акцент и акцентируемый символ были из различных шрифтов.) Если акцент должен передвигаться вверх или вниз, он помещается в h-бокс, который поднимается или опускается. Затем акцент накладывается на символ при помощи кернов таким образом, что ширина акцента не влияет на ширину результирующего горизонтального списка. И, наконец, TEX устанавливает `\spacefactor` равным 1000.

■ `\/`. Если последний элемент текущего списка — символ или лигатура, то добавляется явный керн для его курсивной поправки.

■ `\discretionary`⟨общий текст⟩⟨общий текст⟩⟨общий текст⟩. В этой команде общие тексты обрабатываются в частной горизонтальной моде. Они должны содержать элементы только фиксированной ширины, следовательно, в этом смысле они не очень общие. Более того, горизонтальный список, сформированный каждым отпадающим общим текстом, должен состоять только из символов, лигатур, кернов, боксов и линеек. В нем не должно быть клея, штрафов и т. д. Эта команда добавляет отпадающий элемент к текущему списку (см. главу 14 для значения отпадающего элемента). Коэффициент пробела не меняется.

■ `\-`. Эта команда эквивалентна `\discretionary{-}{}{}`.

■ `$`. Символ "математического переключателя" указывает TEX'у войти в математическую или выделенную математическую моду следующим образом: TEX смотрит на следующий элемент, не раскрывая его. Если этим элементом является `$` и если TEX в это время находится в горизонтальной моде, то TEX разбивает текущий абзац на строки, как объяснялось выше (если только текущий список не пуст), возвращаясь в охватывающую вертикальную или внутреннюю вертикальную моду, вычисляет такие значения, как `\prevgraf`, `\displaywidth` и `\predisplaysize`, входит на новый уровень группирования, вставляет во входную информацию элементы `\everydisplay`, выполняет функции планировщика страницы, обрабатыва-

ет "⟨математический материал⟩$$" в выделенной математической моде, помещает выделение в охватывающий вертикальный список, как объяснялось в главе 19 (допуская переселение вертикального материала), снова выполняет функции планировщика страницы, увеличивает `\prevgraf` на 3 и снова возобновляет горизонтальную моду с пустым списком и с коэффициентом пробела, равным 1000. (Это понятно?) В противном случае TEX возвращает увиденный элемент назад во входную информацию, входит на новый уровень группирования, вставляет элементы `\everymath` и обрабатывает

"⟨математический материал⟩$". Математический материал преобразуется в горизонтальный список и добавляется к текущему списку, окруженный элементами "включить математику" и "выключить математику", а коэффициент пробела устанавливается в 1000. Одно из следствий этих правил — то, что $$ в частной горизонтальной моде дает просто пустую математическую формулу.

- То, что выше не указано. Если в горизонтальной моде встретится любая другая примитивная команда TEX'а, будет дано сообщение об ошибке, и TEX попробует исправить ее на что-либо разумное. Например, если появится верхний или нижний индекс или будет дана какая-нибудь другая чисто математическая команда, TEX попытается вставить $ перед согрешившим элементом; это введет математическую моду.

Наоборот. *Можно превратить все* Вертикали *в* Горизонтали.

Otherwise. *You may reduce all* Verticals *into* Horizontals.

— JOSEPH MOXON, *A Tutor to Astronomie and Geographie* (1659)

```
! You can't use `\moveleft' in horizontal mode.
```

— TEX (1982)

26

Обзор математической моды

В заключении обзора, который был начат в главе 24, давайте исследуем точно, что может делать пищеварительный процесс TEX'а, когда TEX строит списки в математической или в выделенной математической моде.

* * *

Три звездочки, точно такие же, как напечатаны здесь, есть и недалеко от конца главы 24. Все, что предшествует в главе 24 трем звездочкам, применимо к математической моде с тем же успехом, что и к вертикальной моде, так что нет необходимости повторять все эти правила. В частности, глава 24 разъясняет команды присваивания и рассказывает, как керны, штрафы, метки, вставки, регулировки и "что-то" ("whatsits") помещаются в математические списки. Теперь наша цель рассмотреть команды, которые имеют специфический математический аромат, в том смысле, что в математической моде ведут себя иначе, чем в вертикальных или горизонтальных модах.

Математические списки — это нечто отличающиеся от других списков TEX'а, потому что они содержат трехлучевые "атомы" (см. главу 17). Атомы бывают тринадцати видов Ord, Op, Bin, Rel, Open, Close, Punct, Inner, Over, Under, Acc, Rad и Vcent. Каждый атом состоит из трех "полей", которые называются его ядром, верхним индексом и нижним индексом, и каждое поле либо пусто, либо заполнено математическим символом, боксом или вспомогательным математическим списком. Математические символы, в свою очередь, имеют два компонента: номер семейства и номер позиции.

Удобно ввести несколько дополнительных синтаксических правил, чтобы указать, что входит в математический список:

⟨символ⟩ ⟶ ⟨буква⟩ | ⟨другой символ⟩ | `\char`⟨8-битное число⟩
| ⟨char-определенный элемент⟩
⟨мат.символ⟩ ⟶ `\mathchar`⟨15-битное число⟩
| ⟨mathchar-определенный элемент⟩
| `\delimiter`⟨27-битное число⟩
⟨мат.обозначение⟩ ⟶ ⟨символ⟩ | ⟨мат.символ⟩
⟨мат.поле⟩ ⟶ ⟨мат.обозначение⟩ | ⟨заполнитель⟩`{`⟨материал мат.моды⟩`}`
⟨ограничитель⟩ ⟶ ⟨заполнитель⟩`\delimiter`⟨27-битное число⟩
| ⟨заполнитель⟩⟨буква⟩ | ⟨заполнитель⟩⟨другой символ⟩

Мы уже видели ⟨символ⟩ в главе 25. Конечно, символы — это основная пища TEX'а: подавляющее большинство всех команд, которые достигают пищеварительного процесса TEX'а в горизонтальной моде, являются примерами команд типа ⟨символ⟩, которые задают число от 0 до 255, указывающее TEX'у печатать соответствующий символ текущего шрифта. Когда TEX находится в математической или в выделенной математической моде, команда ⟨символ⟩ принимает дополнительное значение: она задает число между 0 и $32767 = 2^{15} - 1$. Это делается замещением символьного номера на значение `\mathcode`, если символьный номер между 0 и 127; символьный номер в границах от 128 до 255 остается неизменным. Если значение `\mathcode` оказывается равным 32768, тем не менее, ⟨символ⟩ замещается активным символьным элементом, имеющим оригинальный символьный код (от 0 до 127); TEX забывает начальный ⟨символ⟩ и раскрывает этот активный символ в соответствии с правилами главы 20.

⟨Мат.символ⟩ определяет 15-битное число, либо задавая его прямо при помощи `\mathchar` или в предыдущем `\mathchardef`, либо задавая 27-битное значение `\delimiter`. В последнем случае отбрасываются самые младшие значащие 12 бит.

Отсюда следует, что каждый ⟨мат.символ⟩, как определено синтаксисом выше, задает 15-битное число, то есть число между 0 и 32767. Такое число может быть представлено в виде $4096c + 256f + a$, где $0 \le c < 8$, $0 \le f < 16$ и $0 \le a < 256$. Если $c = 7$, TEX изменяет c на 0; если текущее значение `\fam` находится между 0 и 15, замещает f на `\fam`. Эта процедура дает во всех случаях номер класса c от 0 до 6, номер семейства f от 0 до 15 и номер позиции a от 0 до 255. (TEX задает начальное значение `\fam`, неявно помещая присваивание `\fam=-1` в начале `\everymath` и `\everydisplay`. Таким образом, подстановка `\fam` вместо f произойдет, только если пользователь явно изменил `\fam` внутри формулы.)

⟨Мат.поле⟩ используется, чтобы задать ядро, верхний и нижний индексы атома. Когда ⟨мат.поле⟩ представляет собой ⟨мат.обозначение⟩, числа f и a этого обозначения отправляются в атомное поле. В противном случае ⟨мат.поле⟩ начинается с `{`, которая указывает TEX'у войти на новый уровень группирования и начать новый математический список; ⟨материал мат.моды⟩ оканчивается знаком `}`, в этой точке завершается группа и результирующий математический список отправляется в атомное поле. Если математический список оказывается просто обычным атомом Ord без верхних и нижних индексов, то охватывающие фигурные скобки убираются.

⟨Ограничитель⟩ используется, чтобы определить как "маленький символ" a в семействе f, так и "большой символ" b в семействе g, где $0 \le a, b \le 255$, а $0 \le f, g \le 15$. Эти символьные коды используются, чтобы создать ограничители переменного размера, как объясняется в приложении G. Если ⟨ограничитель⟩ задан явно в терминах 27-битного числа, то желаемые коды получаются, если интерпретировать это число как $c{\cdot}2^{24}+f{\cdot}2^{20}+a{\cdot}2^{12}+g{\cdot}2^{8}+b$, игнорируя значение c. В противном случае ограничитель задается как ⟨буква⟩ или ⟨другой символ⟩, а 24-битное значение `\delcode` этого символа интерпретируется как $f \cdot 2^{20} + a \cdot 2^{12} + g \cdot 2^8 + b$.

Теперь давайте изучать конкретные команды и то, как TEX подчиняется им в математической моде. Рассмотрим сначала те из них, которые имеют аналоги в вертикальной и/или в горизонтальной моде:

- `\hskip`⟨клей⟩, `\hfil`, `\hfill`, `\hss`, `\hfilneg`, `\mskip`⟨mu-клей⟩. Клей добавляется к текущему математическому списку.

- ⟨проводники⟩⟨бокс или линейка⟩⟨мат.пропуск⟩. ⟨Мат.пропуск⟩ — это одна из шести только что упомянутых команд, которые добавляют клей. Формальный синтаксис для ⟨проводники⟩ и ⟨бокс или линейка⟩ дан в главе 24. Клей, полученный проводниками, добавляется к текущему списку.

- `\nonscript`. Добавляется специальный клей, ширина которого равна нулю. Это действует, как удаление из списка следующего элемента, если этот элемент — клей и если `\nonscript` набирается в "стиле индекса" или в "стиле повторного индекса."

- ⟨элемент пробела⟩. Пробелы в математических модах не действуют.

- `\␣`. Командный пробел добавляет клей к текущему списку, используя ту

же самую величину, которую ⟨элемент пробела⟩ вставляет в горизонтальной моде, когда коэффициент пробела равен 1000.

■ ⟨бокс⟩. Конструируется бокс, и, если результат пустой, ничего не происходит. В противном случае к математическому списку добавляется новый атом типа Ord, и бокс становится его ядром.

■ `\raise`⟨размер⟩⟨бокс⟩, `\lower`⟨размер⟩⟨бокс⟩. Это действует точно так же, как обычная команда ⟨бокс⟩, но новый бокс, который помещается в ядро, сдвигается вверх или вниз на указанную величину.

■ `\vcenter`⟨спецификация бокса⟩`{`⟨материал вертикальной моды⟩`}`. Формируется v-бокс, как если бы вместо `\vcenter` был `\vbox`. Затем новый Vcent-атом добавляется к текущему математическому списку и бокс становится его ядром.

■ ⟨вертикальная линейка⟩. Линейка добавляется к текущему списку (не как атом).

■ `\halign`⟨спецификация бокса⟩`{`⟨материал выравнивания⟩`}`. Эта команда разрешена только в выделенной математической моде и только когда текущий математический список пустой. Выравнивание обрабатывается точно так же, как если бы оно делалось в охватывающей вертикальной моде (см. главу 24), за исключением того, что строки сдвинуты вправо на `\displayindent`. За закрывающей фигурной скобкой могут следовать необязательные команды ⟨присваивание⟩, после которых `$$` должно закончить выделение. TEX вставит `\abovedisplayskip` и `\belowdisplayskip` клей перед и после результата выравнивания.

■ `\indent`. К списку добавляется пустой бокс шириной `\parindent` как ядро нового атома Ord.

■ `\noindent`. Эта команда в математической моде не действует.

■ `{`⟨материал мат.моды⟩`}`. Символьный элемент категории 1 или команда типа `\bgroup`, которая сделана `\let`-равной такому символьному элементу, указывает TEX'у начать новый уровень группирования, а также начать обрабатывать новый математический список. Когда такая группа заканчивается — при помощи `}` — TEX использует полученный математический список как ядро нового атома Ord, который добавляется к текущему списку. Следовательно, если результирующий математический список — это простой атом Acc (т.е., акцентированная величина), то добавляется сам этот атом.

■ ⟨мат.обозначение⟩. (Это наиболее общая команда в математической моде; смотрите ее синтаксис ближе к началу этой главы.) Математический символ, как объяснялось ранее, определяет три величины c, f и a. TEX добавляет атом к текущему списку, где атом имеет тип Ord, Op, Bin, Rel, Open, Close или Punct, в соответствии с тем, чему равно c, 0, 1, 2, 3, 4, 5 или 6. Ядро такого атома — это математическое обозначение, определенное величинами f и a.

■ ⟨мат.атом⟩⟨мат.поле⟩. Команда ⟨мат.атом⟩ — это одно из следующего:

```
\mathord | \mathop | \mathbin | \mathrel | \mathopen
    | \mathclose | \mathpunct | \mathinner | \underline | \overline
```

TEX обрабатывает ⟨мат.поле⟩, затем добавляет новый атом заданного типа к текущему списку. Ядро этого атома содержит заданное поле.

■ `\mathaccent`⟨15-битное число⟩⟨мат.поле⟩. TEX преобразует число в c, f и a, как он это делал с `\mathchar`. Затем он обрабатывает ⟨мат.поле⟩ и добавляет новый Acc-атом к текущему списку. Ядро этого атома содержит указанное поле; символ акцента в этом атоме содержит (a, f).

■ `\radical`⟨27-битное число⟩⟨мат.поле⟩. TEX преобразует число в a, f, b и g, как он это делает с любым `\delimiter`. Затем он обрабатывает ⟨мат.поле⟩ и добавляет новый Rad-атом к текущему списку. Ядро этого атома содержит заданное поле; поле ограничителя содержит (a, f) и (b, g).

■ ⟨верхний индекс⟩⟨мат.поле⟩. Команда ⟨верхний индекс⟩ является явным или неявным символьным элементом категории 7. Если текущий список не оканчивается атомом, то добавляется новый Ord-атом с полями, которые все пусты. Таким образом, текущий список во всех случаях будет оканчиваться атомом. Поле верхнего индекса этого атома должно быть пустым; оно становится непустым при изменении его на результат заданного значения ⟨мат.поле⟩.

■ ⟨нижний индекс⟩⟨мат.поле⟩. Команда ⟨нижний индекс⟩ — это явный или неявный символьный элемент категории 8. Она действует точно так же, как команда ⟨верхний индекс⟩, за исключением, конечно, того, что действует на поле нижнего индекса вместо поля верхнего индекса.

■ `\displaylimits`, `\limits`, `\nolimits`. Эти команды разрешены, только если текущий список оканчивается атомом Op. Они изменяют специальное поле этого атома Op, указывая, какие соглашения должны быть использованы относительно пределов. Нормальное значение этого поля — `\displaylimits`.

■ `\/`. К текущему списку добавляется керн нулевой ширины. (Это будет действовать, как добавление курсивной поправки к предыдущему символу, если курсивная поправка не была добавлена обычным способом.)

■ `\discretionary`⟨общий текст⟩⟨общий текст⟩⟨общий текст⟩. Эта команда трактуется точно также, как в горизонтальной моде (см. главу 25), но третий ⟨общий текст⟩ должен произвести пустой список.

■ `\-`. Эта команда эквивалентна `\discretionary{-}{}{}`, поэтому "`-`" интерпретируется как дефис, а не как знак минус.

■ `\mathchoice`⟨общий текст⟩⟨общий текст⟩⟨общий текст⟩⟨общий текст⟩. Каждый из четырех общих текстов трактуется как подформула (то есть, как вторая альтернатива в определении ⟨мат.поле⟩). Четыре математических списка, определенные таким образом, записываются в "выбор", который добавляется к текущему списку.

■ `\displaystyle`, `\textstyle`, `\scriptstyle`, `\scriptscriptstyle`. Элемент изменения стиля, который соответствует заданному стилю, добавляется к текущему списку.

■ `\left`⟨ограничитель⟩⟨материал мат.моды⟩`\right`⟨ограничитель⟩. В этом случае TEX начинает новую группу и обрабатывает ⟨материал мат.моды⟩, образовывая новый математический список, который начинается с левого граничного элемента, содержащего первый ограничитель. Эта группа должна прерваться командой `\right`, и в это время внутренний математический список завершается правым граничным элементом, содержащим второй ограничитель. Затем TEX добавляет атом Inner к текущему списку; ядро этого атома содержит внутренний математический список.

■ ⟨команда обобщенной дроби⟩. Эта команда имеет одну из шести форм:

```
\over | \atop | \above⟨размер⟩
      | \overwithdelims⟨ограничитель⟩⟨ограничитель⟩
      | \atopwithdelims⟨ограничитель⟩⟨ограничитель⟩
      | \abovewithdelims⟨ограничитель⟩⟨ограничитель⟩⟨размер⟩
```

(см. главу 17). Когда TEX'у встречается ⟨команда обобщенной дроби⟩, он берет элемент текущего списка и помещает его в поле числителя элемента обобщенной дроби. Поле знаменателя в этом новом элементе временно является пустым; поля левого и правого ограничителя устанавливаются равными заданным кодам ограничителей. TEX хранит этот элемент обобщенной дроби в специальном месте, связанным с текущим уровнем выполнения математической моды. (В этом специальном месте не должно быть другого элемента обобщенной дроби, поскольку конструкции типа `a\over b\over c` незаконны). Затем TEX делает текущий список пустым и продолжает обрабатывать команды в математической моде. Позднее, когда текущий уровень математической моды завершен (либо встретившимся `$`, либо `}`, либо `\right`, в зависимости от природы текущей группы), текущий список будет помещен в знаменатель элемента обобщенной дроби, которая была сохранена; затем этот элемент весь целиком займет место в полном списке. Однако в специальном случае, в котором текущий список начинается `\left` и заканчивается `\right`, граничные элементы будут извлечены из числителя и знаменателя обобщенной дроби, и конечный список будет состоять из трех элементов: левая граница, обобщенная дробь, правая граница. (Если вы хотите проследить процесс, который строит математические списки, для вас может оказаться полезным ввести `\showlists` в то время, когда TEX обрабатывает знаменатель обобщенной дроби.)

■ ⟨номер уравнения⟩⟨материал мат.моды⟩`$`. Здесь ⟨номер уравнения⟩ — это либо `\eqno` или `\leqno`. Эти команды разрешены только в выделенной математической моде. Когда TEX прочитывает ⟨номер уравнения⟩, он входит на новый уровень группирования, вставляет символы `\everymath` и входит в невыделенную математическую моду, чтобы поместить ⟨материал мат.моды⟩ в математический список. Когда этот математический список завершен, TEX преобразует его в горизонтальный список и помещает результат в бокс, который будет использован как номер уравнения в текущем выделении. Закрывающий знак `$` будет помещен назад во входной поток, где он будет прерывать выделение.

■ `$`. Если TEX находится в выделенной математической моде, он читает еще один знак, который также должен быть `$`. В любом случае, команда математического сдвига прерывает текущий уровень обработки математической моды и заканчивает текущую группу, которую должен был начать либо `$`, либо ⟨номер уравнения⟩. Как только математический список окончен, он преобразуется в горизонтальный список, как объяснялось в приложении G.

■ Нечто, что не указано выше. Если в математической моде встречается любая другая примитивная команда TEX'а, будет дано сообщение об ошибке, и TEX попробует исправить ее на что-либо разумное. Например, если появляется команда `\par` или дается любая другая существенно нематематическая команда, TEX попытается вставить `$` непосредственно перед согрешившим знаком. Это выводит из математической моды. С другой стороны, если в математической моде

появляется совсем перепутанный знак типа `\endcsname`, `\omit` или `#`, TEX будет его просто игнорировать после сообщения об ошибке. Вы можете поразвлечься, вводя какую-нибудь совсем глупую информацию, чтобы посмотреть, что получится. (Сначала скажите `\tracingall`, как объяснялось в главе 27, чтобы получить максимум информации).

Упражнение 26.1
Степени десятки. Сейчас сделан полный обзор всего языка TEX. Для того, чтобы продемонстрировать, насколько много вы знаете, назовите все случаи, которые можете придумать, когда числа 10, 100, 1000, 10000 и 100000 имеют в TEX'е специальное значение.

Упражнение 26.2
Степени двойки. Назовите все случаи, которые можете придумать, когда числа 8, 16, 32, 64, 128, 256, ... имеют в TEX'е специальное значение.

Профессионалы считают набор математики трудным *или* штрафным, *ибо набор ее медленнее, труднее и дороже, чем любой другой набор, обычно встречающийся в книгах и журналах.*

Mathematics is known in the trade as difficult, *or* penalty, copy *because it is slower, more difficult, and more expensive to set in type than any other kind of copy normally occurring in books and journals.*

— UNIVERSITY OF CHICAGO PRESS, *A Manual of Style* (1969)

Сложна сказка Математики, и она не поддается ни краткому изложению сюжета, ни точной формулировке ее морали.

The tale of Math is a complex one, and it resists both a simple plot summary and a concise statement of its meaning.

— PATRICK K. FORD, *The Mabinogi* (1977)

27

Исправление ошибок

О'кей, все, что вам надо знать о TEX'е, уже объяснено, если, конечно, вам везет и вы не делаете ошибок. Если вы и дальше собираетесь не делать никаких ошибок, не затрудняйтесь читать эту главу. В противном случае может оказаться полезным использовать некоторые способы, которыми TEX пытается обнаружить изъяны вашей рукописи.

В экспериментальном запуске, который вы делали, когда читали Главу 6, вы изучали общую вид сообщений об ошибке, а также различные способы, которыми можно отвечать на жалобы TEX'а. Со временем вы сможете исправлять большинство ошибок "online" по мере того, как TEX обнаруживает их, вставляя или удаляя что-либо. Правильный подход к этому — общаться с TEX'ом в хорошем настроении и рассматривать полученные сообщения об ошибках как забавные головоломки — "Почему машина это сделала?" — а не как личные оскорбления.

TEX умеет выдавать более сотни различных видов сообщений об ошибках, и вы, вероятно, никогда не столкнетесь со всеми из них, поскольку некоторые типы ошибок очень трудно сделать. В главе 6 мы обсудили ошибку "неопределенная команда". Теперь давайте посмотрим на некоторые другие.

Если вы ошиблись в названии некоторой единицы измерения — например, если вы ввели "`\hsize=4im`" вместо "`\hsize=4in`" — то получите сообщение об ошибке, которое выглядит как-то так:

```
! Illegal unit of measure (pt inserted).
<to be read again>
                   i
<to be read again>
                   m
<*> \hsize=4im
               \input story
?
```

TEX должен найти правильную единицу, прежде чем продолжать работу, так что в этом случае он неявно вставляет "`pt`" в текущую позицию входных данных и устанавливает `\hsize=4pt`.

Как лучше исправить такую ошибку? Ну, если вы не уверены, что означает такое сообщение, надо ввести `H` или `h`, чтобы увидеть вспомогательное сообщение. Затем можно прочитать это сообщение и увидеть, что если просто нажать ⟨return⟩ и продолжить работу, TEX прочтет "`i`", затем "`m`", а затем `\input story`. К несчастью, это легкое решение не самое лучшее, поскольку "`i`" и "`m`" будут напечатаны в новом абзаце. В этом случае возможно гораздо более изящное исправление. Сначала надо ввести "`2`". Это говорит TEX'у удалить следующие два символа, которые он читает. После того, как TEX сделает это, он опять остановится для того, чтобы дать вам

шанс выйти на новую ситуацию. И вот что вы увидите:

```
<recently read> m

<*> \hsize=4im
                \input story
?
```

Хорошо, "i" и "m" прочитаны и выброшены. Но если вы сейчас нажмете ⟨return⟩, TEX выполнит команду `\input story` и попытается напечатать файл `story.tex` с `\hsize=4pt`, что будет не особенно волнующим экспериментом, поскольку просто даст десятки переполненных боксов, по одному для каждого звука рассказа (story). И опять есть более хороший путь: можно вставить команду, которую вы и подразумевали, введя

```
I\hsize=4in
```

Это дает TEX'у задание изменить `\hsize` на правильное значение, после чего последует `\input story`, и вы спокойно пойдете дальше.

▸ **Упражнение 27.1**
Ben User напечатал "8", а не "2" в ответ на только что рассмотренное сообщение об ошибке: ему хотелось удалить "i", "m", "`\input`" и еще пять букв "`story`". Но TEX ответил

```
<*> \hsize=4im \input stor
                           y
```

Объясните, что случилось.

TEX обычно старается исправить ошибки, либо игнорируя команду, которую не понимает, либо вставляя что-нибудь, что сохранит его покой. Например, в главе 6 вы видели, что TEX игнорирует неопределенную команду, а только что наблюдали, как TEX вставляет "`pt`", когда ему нужна физическая единица измерения. Приведем другой пример, когда TEX что-нибудь вставляет:

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   ^
l.11 Тот факт, что 32768=2^
                          {15} не очень интересен
? H
I've inserted a begin-math/end-math symbol since I think
you left one out. Proceed, with fingers crossed.
```

(Пользователь забыл заключить формулу в знаки `$`, и TEX попытался исправить это, вставив один из них.) В этом случае явно показан вставляемый

текст, который еще не прочитан. Наоборот, наш предыдущий пример иллюстрировал случай, когда TEX уже ввел "`pt`", который он вставил. Таким образом, здесь пользователь имеет шанс убрать вставленный `$` перед тем, как TEX осознает его.

Что надо делать? Ошибка в этом примере встретилась до того, как TEX заметил что-то неправильное. Символы "`32768=2`" уже набраны в горизонтальной моде. Нет способа вернуться назад и удалить прошлое, так что отсутствие подходящих пробелов вокруг `=` не может быть замечено. Поэтому сейчас при исправлении ошибки мы хотим не получить безукоризненный результат, а только продолжить как-нибудь работу, чтобы TEX пропустил эту ошибку и обнаружил следующую. Если бы мы сейчас просто нажали ⟨return⟩, то получили бы не то, что хотели, поскольку TEX напечатал бы последующий текст, как математическую формулу: "^{15}неоченьинтересен..." и обнаружил бы другую ошибку, когда абзац кончился до закрывающего `$`. Существует более совершенный способ исправления, а именно, ввести "`6`", а затем "`I$^{15}$`". Это удаляет "`$^{15}`" и вставляет правильную часть формулы. Но это неоправдано сложно. Самым лучшим решением будет ввести только "`2`", а затем продолжить работу. TEX будет печатать неправильное выражение "32768=215", но вы сможете проверить остаток текста, как если бы эта ошибка не встретилась.

▸ **Упражнение 27.2**

Вот пример, в котором была нечаянно пропущена обратная косая черта:

```
! Missing control sequence inserted.
<inserted text>
                \inaccessible
<to be read again>
                   m
l.10 \def m
           acro{replacement}
```

TEX'у после "`\def`" нужна команда, поэтому он вставляет то, что позволяет продолжать работу. (Здесь показана команда "`\inaccessible`" ("недоступная"), но это не имеет отношения к той команде, которую вы хотели задать в своей рукописи.) Если вы в этом месте просто нажмете ⟨return⟩, TEX сам определит недоступную команду, но в этом нет ничего хорошего: будет не определена более поздняя ссылка на `\macro`. Объясните, как избавиться от такой ошибки, чтобы строка 10 входного файла говорила "`\def\macro{replacement}`".

▸ **Упражнение 27.3**

Когда вы, чтобы ответить на сообщение об ошибке, используете опцию "`I`", правила главы 8 предполагают, что TEX удаляет все пробелы на правом конце вставляемой строки. Объясните, как можно использовать опцию "`I`", чтобы, несмотря на это, вставить пробел.

Труднее всего справляться с такими ошибками, при которых вы ошиблись, скажем, на строке 20, а TEX ничего не говорит, пока не доберется,

скажем, до строки 25. Например, если вы забыли "}", которая завершает аргумент некоторой макрокоманды, TEX не заметит никакой проблемы, пока не достигнет конца абзаца. Но обычно TEX сможет начать выполнение следующего абзаца, как если бы ничего не случилось. "Убежавший аргумент" будет показан на дисплее и просматривая этот текст, вы должны будете вычислить, где была пропущена "}".

Полезно помнить, что первая ошибка в вашем документе может в дальнейшем породить фальшивые "ошибки", поскольку ненормальные команды могут нанести серьезный удар по способности TEX'а набирать последующий материал. Но в большинстве случаев оказывается, что простой прогон через машину отметит все места, в которых ваши входные данные противоречат правилам TEX'а.

Ситуация становится гораздо более серьезной, когда ваша ошибка вызвана путаницей в понимании, а не опечаткой: тут сообщение об ошибке, вероятно, будет не очень полезным, даже если вы попросите TEX помочь вам. Если вы без должного понимания переопределили команду — например, сказали `\def\box{...}`, могут произойти все виды несчастий. Компьютер не ясновидец, и TEX может только объяснить, что выглядит неправильно с его точки зрения. Эти объяснения кажутся таинственными, если вы не можете понять позицию машины. В таких случаях, безусловно, надо обратиться к кому-нибудь за советом или консультацией или, в крайнем случае, прочитать инструкции в главах 2, 3, ..., 26.

▸ **Упражнение 27.4**

J. H. Quick (студент) однажды определил следующий набор макрокоманд:

```
\newcount\serialnumber
\def\firstnumber{\serialnumber=0 }
\def\nextnumber{\advance \serialnumber by 1
  \number\serialnumber)\nobreak\hskip.2em }
```

Таким образом, он мог вводить, например,

```
\firstnumber
\nextnumber xx, \nextnumber yy и \nextnumber zz
```

и TEX печатал "1) xx, 2) yy и 3) zz". Это прекрасно работало, и он показал макрокоманды своим приятелям. Но несколькими месяцами позже он получил неистовый телефонный звонок: один из его друзей натолкнулся на совершенно роковое сообщение об ошибке:

```
! Missing number, treated as zero.
<to be read again>
                   c
l.107 \nextnumber minusc
                        ule chances of error
?
```

Объясните, что случилось, и посоветуйте ему, что делать.

Раньше или позже — надеемся, что раньше — TEX обработает весь ваш файл без единой жалобы. Но может оказаться, что результат все еще не правильный. Тот факт, что TEX не останавливался, еще не означает, что можно обойтись без корректировки. На этой стадии обычно легко разделаться с опечатками, исправляя входные данные. Ошибки макета можно устранить, используя методы, которые мы обсуждали прежде, переполненные боксы вылечить, как описывалось в Главе 6, плохого разбиения можно избежать, используя связки или команды `\hbox`, как объяснялось в главе 14, а математические формулы улучшить, применяя принципы глав 16 – 19.

И все-таки результат может все еще содержать на первый взгляд необъяснимые ошибки. Например, если вы указали шрифт с некоторым увеличением, которое не поддерживается вашим математическим обеспечением, TEX не узнает, что здесь возникает ошибка, а программа, которая преобразует `dvi`-файл в физическую копию, может и не сказать, что заменила вам шрифт на "приблизительно похожий". Полученное распределение пробелов может выглядеть совершенно ужасным.

Если вы не можете найти, в чем причина ошибок, попробуйте старый трюк упрощения вашей программы: убирайте все, что работает, пока не получите наиболее короткий входной файл, который ошибается так же, как оригинал. Чем короче файл, тем легче будет вам или кому-нибудь еще засечь ошибку.

Возможно, вы удивитесь, почему TEX не поместил пробел там, где, по вашему мнению, должен. Помните, что когда TEX читает ваш файл, он игнорирует пробелы, которые следуют за командными словами. (TEX также игнорирует пробелы после числа или единицы измерения, которые появляются в качестве аргумента в примитивной команде. Но если вы используете правильно сконструированные макрокоманды, эти правила вас не касаются, поскольку вы не используете примитивные команды непосредственно.)

Но если вы сами создаете макрокоманды, задача аварийного ремонта может быть намного более сложной. Например, вы можете обнаружить, что TEX испустил три пробела, когда обрабатывал некоторую длинную последовательность сложных кодов, состоящую из нескольких десятков команд. Как найти, откуда заползли эти пробелы? Надо установить `\tracingcommands=1`, как упоминалось в главе 13. Это говорит TEX'у делать запись в протокольном файле всякий раз, когда он начинает выполнять примитивную команду, и вы сможете увидеть, когда командой будет "`blank space`".

Большинство реализаций TEX'а позволяет некоторым образом прерывать программу. Это дает возможность диагностировать причины бесконечных циклов. TEX при прерывании переходит в `\errorstopmode`, следовательно, появляется возможность вставить команды во входные данные: можно оборвать прогон, сделать `\show`, изменить текущее содержание команд, регистров и так далее. Можно также понять, на что TEX тратит больше всего времени, если вы используете непроизводительную макрокоманду, поскольку беспорядочные прерывания чаще появляются в том месте, которое TEX посещает наиболее часто.

Иногда ошибка настолько серьезна, что TEX вынужден преждевременно закончить работу. Например, если вы работаете в `\batchmode` или `\nonstopmode`, TEX делает "аварийную остановку", если ему нужен ввод с терминала. Это случается, когда нужный файл не может быть открыт или когда во входном документе не найдена команда `\end`. Приведем некоторые послания, которые вы можете получить непосредственно перед тем, как TEX испустит дух:

```
Fatal format error;I'm stymied.
```

("Фатальная ошибка формата: я в безвыходном положении.") Это означает, что указанный вами предварительно загруженный формат не может быть использован, поскольку он был подготовлен для другой версии TEX'а.

```
That makes 100 errors; please try again.
```

(Сделано 100 ошибок; пожалуйста, попробуйте снова.) TEX записал уже 100 ошибок с тех пор, как кончился последний абзац, так что он, возможно, находится в бесконечном цикле.

```
Interwoven alignment preambles are not allowed.
```

(Переплетенные преамбулы выравниваний не разрешены.) Если вы настолько хитры, чтобы получить такое сообщение, то поймете его, и не заслуживаете никаких симпатий.

```
I can't go on meeting you like this.
```

(Я не могу продолжать, встретив такое.) Предыдущая ошибка нанесла TEX'у сильный удар. Устраните ее и попробуйте снова.

```
This can't happen.
```

(Этого не может случится.) Что-то неправильное с TEX'ом, который вы используете. Отчаянная жалоба.

Существует также ужасное послание, которое TEX выпускает с огромной неохотой. Но оно может встретиться:

```
TeX capacity exceeded, sorry.
```

(Вместимость TeX'а превышена, извините.) Это, увы, означает, что вы пытались растянуть TEX слишком сильно. Такое сообщение говорит вам, что часть памяти TEX'а перегружена. Будет упомянуто одно из следующих четырнадцати наименований:

`number of strings` (имена команд и файлов)
`pool size` (символы в таких именах)
`main memory size` (боксы, клей, точки разрыва и т.д.)
`hash size` (имена команд)
`font memory` (метрические данные шрифта)
`exception dictionary` (исключения для переноса)
`input stack size` (одновременные входные потоки)
`semantic nest size` (конструируются незаконченные списки)
`parameter stack size` (макропараметры)
`buffer size` (символы в строках читаются из файла)
`save size` (значение для восстановления по концу группы)

```
text input levels     (\input-файлы и ошибочные вставки)
grouping levels     (незаконченные группы)
pattern memory     (данные образцов переноса)
```

Текущая величина доступной памяти будет также показана.

Если вы не переполняете возможности TEX'а, а просто хотите увидеть, насколько близко вы приблизились к их пределам, установите перед концом работы `\tracingstats` в положительное значение. Тогда протокольный файл будет завершаться отчетом о том, как использовались первые одиннадцать из названных выше наименований (т.е., `number of strings`, ..., `save size`), в том же порядке. Более того, если вы установите `\tracingstats` равным 2 или более, TEX будет показывать текущее использование памяти всякий раз, когда выполняет команду `\shipout`. Такие статистические данные разбиваются на две части; "`490&5950`" означает, например, что 490 слов используется для "больших" вещей, таких, как боксы, клей, точки разбиения, а 5950 слов используется для "маленьких" вещей, таких, как элементы и символы.

Что можно сделать, если превышена вместимость TEX'а? Все перечисленные выше компоненты вместимости могут быть увеличены, при условии, что у вас достаточно большой компьютер. Действительно, место для увеличения одной компоненты обычно можно получить, уменьшая некоторую другую компоненту и не увеличивая общий размер памяти TEX'а. Если у вас особо важное задание, вы, быть может, сможете убедить местных системщиков обеспечить вас специальным TEX'ом, вместимости которого специально приспособлены для ваших нужд. Но перед тем, как предпринять такой решительный шаг, удостоверьтесь, что вы используете TEX должным образом. Если задан гигантский абзац или гигантское выравнивание, которое охватывает более одной страницы, то измените свой подход, поскольку TEX прочитывает все подряд до конца, перед тем как начинает разбивать строки или вычислять выравнивание, а это поглощает огромную величину памяти. Если у вас необычная макробиблиотека, не забывайте, что TEX должен помнить все тексты замены, которые вы определяете, поэтому, если размер памяти дефицитен, надо загружать только те макрокоманды, которые вам действительно нужны. (См. в приложениях B и Dидеи, как сделать макроопределения более компактными.)

Некоторые ошибочные программы TEX'а будут переполнять любой конечный объем памяти. Например, после `\def\recurse{(\recurse)}` и команды `\recurse` TEX немедленно затараторит:

```
! TeX capacity exceeded, sorry [input stack size=80].
\recurse ->(\recurse
                    )
\recurse ->(\recurse
                    )
...
```

Очевидно, та же ошибка будет появляться независимо от того, насколько вы увеличили размер входного стэка TEX'а.

Превышения вместимости "`save size`" ("размера спасения") является одной из самых мучительных для исправления ошибок, особенно если вы

наталкиваетесь на эту ошибку только при длинных работах. TEX обычно тратит по два слова размера спасения каждый раз, когда выполняет неглобальное присваивание некоторой величине, предыдущее значение которой не было присвоено на том же уровне группирования. Когда макрокоманды написаны грамотно, в "save stack" редко нужно место для более чем 100 таких вещей. Но можно безгранично увеличить запрос на "save stack", если делать как локальные, так и глобальные присваивания одной и той же переменной. Можно посмотреть, что TEX помещает в "save stack", установив `\tracingrestores=1`. Ваш протокольный файл будет содержать информацию обо всем, что передвигается из стэка в конец группы. Например, пусть `\a` обозначает команду `\advance\day by 1`, а `\g`— команду `\global\advance\day by 1`. Рассмотрим следующие команды:

```
\day=1 {\a\g\a\g\a}
```

Первая `\a` устанавливает `\day=2` и запоминает старое значение `\day=1`, помещая его в стэк спасения. Первая `\g` устанавливает `\day=3` глобально; во время глобального присваивания отправлять в стэк спасения ничего не надо. Следующая `\a` устанавливает `\day=4` и запоминает старое значение `\day=3` в стэке спасения. Затем `\g` устанавливает `\day=5`, а `\a` устанавливает `\day=6` и запоминает `\day=5`. Наконец, `}` указывает TEX'у вернуться назад через стэк спасения. Если в этой точке `\tracingrestores=1`, то протокольный файл получит следующие данные:

```
{restoring \day=5}
{retaining \day=5}
{retaining \day=5}
```

Объяснение: параметр `\day` сначала восстанавливает свое глобальное значение 5. Поскольку это значение глобальное, оно будет сохранено, поэтому другие спасенные значения (`\day=3` и `\day=1`), по существу, игнорируются. Мораль: если вы обнаруживаете, что TEX хранит слишком много значений, значит у вас есть набор макрокоманд, которые могут привести стэк спасения к переполнению при достаточно больших работах. Чтобы предотвратить это, обычно стоит быть последовательным в своих присваиваниях каждой переменной, которую вы используете. Присваивания должны быть либо всегда глобальными, либо всегда локальными.

TEX имеет и другие виды трассирования в дополнение к `\tracingstats` и `\tracingrestores`. Мы уже обсуждали `\tracingcommands` в главах 13 и 20, `\tracingparagraphs` в главе 14, `\tracingpages` в главе 15 и `\tracingmacros` в главе 20. Существует также `\tracinglostchars`, которая, если положительна, указывает TEX'у каждый раз записывать символ, который пропускается, поскольку он не должен появляться в текущем шрифте, и `\tracingoutput`, которая, если положительна, указывает TEX'у показывать на терминале в символической форме содержание каждого бокса, который отправляется в `dvi`-файл. Последняя позволяет увидеть, все ли введено правильно, если вы пытаетесь разобраться, вызвана ли некоторая аномалия TEX'ом или какой-нибудь другой программой, которая влияет на результат работы TEX'а.

Когда TEX показывает бокс в диагностике, количество данных управляется параметрами `\showboxbreadth` и `\showboxdepth`. Первый из них, который начальный TEX устанавливает равным 5, задает максимальное число элементов, показанных на одном уровне; второй, который начальный TEX устанавливает

равным 3, задает самый глубокий уровень. Например, маленький бокс, полным содержанием которого является

```
\hbox(4.30554+1.94444)x21.0, glue set 0.5
.\hbox(4.30554+1.94444)x5.0
..\tenrm g
.\glue 5.0 plus 2.0
.\tenrm | (ligature ---)
```

при `\showboxbreadth=1` и `\showboxdepth=1` будет обозначен следующим образом:

```
\hbox(4.30554+1.94444)x21.0, glue set 0.5
.\hbox(4.30554+1.94444)x5.0 []
.etc.
```

А если вы устанавливаете `\showboxdepth=0`, то получите только верхний уровень:

```
\hbox(4.30554+1.94444)x21.0, glue set 0.5 []
```

(Заметим, что как `[]`, так и `etc.` указывают на то, что данные обрезаны.)

Непустой h-бокс считается "переполненным", если клей в нем не способен сжаться до указанного размера, при условии, что `\hbadness` меньше 100 и что лишняя ширина (после сжимания на максимальную величину) больше `\hfuzz`. Он считается "тесным", если клей сжимается и плохость превышает `\hbadness`. Он "свободный", если клей растягивается и плохость превышает `\hbadness`, но не больше 100; бокс "незаполненный", если клей растягивается и плохость больше и `\hbadness` и 100. Аналогичные примечания применимы и к непустым v-боксам. Как только такие аномалии обнаруживаются, TEX печатает предупреждающее сообщение и показывает согрешивший бокс. Пустые боксы никогда не считаются ненормальными.

Когда выравнивание оказывается "переполненным", "тесным", "свободным" или "незаполненным", вы не получите предупреждающих сообщений для каждой выравниваемой строки, а получите только одно сообщение и *прототипный ряд* (или, при `\valign`, *прототипную колонку*). Например, предположим, вы говорите `\tabskip=0pt plus10pt \halign to200pt{&#\hfil\cr...\cr}`, и оказывается, что выравниваемый материал делает две колонки шириной, соответственно, 50 pt и 60 pt. Тогда вы получите следующее сообщение:

```
Underfull \hbox (badness 2698) in alignment at lines 11--18
 [] []
\hbox(0.0+0.0)x200.0, glue set 3.0
.\glue(\tabskip) 0.0 plus 10.0
.\unsetbox(0.0+0.0)x50.0
.\glue(\tabskip) 0.0 plus 10.0
.\unsetbox(0.0+0.0)x60.0
.\glue(\tabskip) 0.0 plus 10.0
```

"Незаданные боксы" в прототипном ряду показывают ширину конкретных колонок. В этом случае табличный клей растянул свою растяжимость в три раза, чтобы достичь цели в 200 pt, так что бокс является незаполненным. (В соответствии с формулой в главе 14, плохость в этой ситуации равна 2700. TEX в

действительности использует аналогичную, но более эффективную формулу, так что вычисленное им значение плохости равно 2698.) Незаполненной является каждая строка выравнивания, но в предупреждающем сообщении будет показан только прототипный ряд. В переполненных выравниваниях к строкам никогда не добавляются "линейки переполнения".


Команды `\tracing...` помещают весь свой вывод в протокольный файл, если только параметр `\tracingonline` не положителен. В последнем случае все диагностическая информация кроме протокольного файла идет и на терминал. Начальный TEX имеет макрокоманду `\tracingall`, которая устанавливает в максимальную величину трассирование всех видов. Она не только обеспечивает `\tracingcommands`, `\tracingrestores`, `\tracingparagraphs` и так далее, но и устанавливает `\tracingonline=1`, `\showboxbreadth` и `\showboxdepth` в максимально высокие значения, так что будет показано все содержание всех боксов.


Некоторые версии TEX'а модернизированы для увеличения скорости. Они не смотрят на значения параметров `\tracingparagraphs`, `\tracingpages`, `\tracingstats` и `\tracingrestores`, поскольку TEX работает быстрее, когда он не должен поддерживать статистику или хранить таблицы на любую требуемую трассировку. Если вы хотите иметь весь диагностический инструмент TEX'а, вы должны быть уверены, что используете правильную версию.


Если установить `\pausing=1`, TEX даст возможность редактировать каждую строку входа, как только она прочитана из файла. В этом случае можно делать временные заплатки (например, вставлять команды `\show...`) во время исправления ошибок, не меняя действительное содержание файла и можно придержать работу TEX'а на человеческой скорости.

Последний совет. При работе над длинной рукописью лучше за один раз подготавливать только несколько страниц. Заведите файл "гранки" и файл "книга", и вводите ваш текст в файл "гранки" (Поместите управляющую информацию, которая задает ваш основной формат, в начало этого файла. Например, в приложении E приведен файл `galley.tex`.) После того, как гранки выглядят правильно, можно добавить их к книжному файлу. Время от времени можно пропускать книжный файл через TEX, чтобы посмотреть, как страницы стыкуются друг с другом. Например, когда автор готовил это руководство, он за один раз делал по одной главе, а длинные главы разделял на подглавы.


▸ **Упражнение 27.5**
Последнее упражнение. Найдите в этом руководстве все обманы и все шутки.

Последний призыв. Вперед к созданию *шедевров издательского искусства!*

Кто может понять свои ошибки? *Who can understand his errors?*
— *Psalm 19 : 12*(c. 1000 B.C.)

Одно дело показать человеку,
что он ошибается,
а другое — наставить его
на путь истинный.

It is one thing, to shew
a Man that he is in an Error,
and another, to put him
in possession of Truth.
— JOHN LOCKE,*An Essay Concerning Humane Understanding* (1690)

A

Ответы на все упражнения

Предисловие к этому рукотводству отмечает, что умнее попробовать выполнить каждое упражнение, прежде чем искать ответ здесь. Но все же эти ответы рассчитаны на то, что вы их прочтете, поскольку подчас дают дополнительную инфомацию, к которой вы будете лучше подготовлены, когда сами поработаете над проблемой.

1.1. TEXник (низкооплачиваемый), иногда также называемый TEXнарь.

2.1. `Алиса сказала: ``Я всегда использую en-тире вместо дефиса, когда указываю в библиографии такие номера страниц: `480--491'.'' (Будет ошибкой в ответе на этот вопрос поставить в конце '$480-491$'.")`

2.2. Вы получите em-тире и дефис (—-), что выглядит ужасно.

2.3. fluffier firefly fisticuffs, flagstaff fireproofing, chiffchaff and riffraff.

2.4. ``\thinspace`; и либо `{}`` или {`}``, либо что-нибудь аналогичное. Причина: обычно пробел *перед* простой левой кавычкой, меньше пробела перед двойной левой кавычкой. (Лево и право являются противоположностями.)

2.5. Исключение `\thinspace` означало бы, что пользователю не нужно учить этот термин. Но не рекомендуется минимизировать терминологию, "перегружая" математическую моду сложными конструкциями. Например, пользователю, который хотел "обмануть" `\mathsurround`, помешало бы нематематическое использование знаков доллара. (Между прочим, ни `\thinspace`, ни `\,` не встроены в TEX. Оба они определены в терминах более примитивных команд в приложении B.)

3.1. `\I`, `\exercise` и `\\`. (Последняя из них имеет тип 2, т.е., командный символ, поскольку вторая обратная косая черта не является буквой. Первая обратная косая черта не дает второй начать свою собственную команду.)

3.2. `math\'ematique` и `centim\`etre`.

3.3. В соответствии с приложением I, `\␣` — это примитив, а \⟨return⟩ нет. В приложении B \⟨return⟩ определяется так: `\def\^^M{\ }`, поскольку возврат представляется как `^^M`. На запрос `\show\^^M` TEX отвечает: "`> \^^M=macro:->\␣`.".

3.4. Существует 128 команд длины 2, и большинство из них в начале работы TEX'а не определены. (TEX позволяет, чтобы функцию сигнального символа выполнял любой символ, и не видит различия между командами, которые предваряются различными сигнальными символами.) Если мы допускаем, что у нас 52 буквы, то существует в точности 52^2 различные команды длины 3 (по одной на каждую пару букв от `AA` до `zz`). Но глава 7 объясняет, как использовать `\catcode` чтобы превратить любой символ в "букву", поэтому можно использовать 128^2 потенциальных команд длины 3.

4.1.
```
Ulrich Dieter, {\sl Journal f\"ur die reine und angewandte
    Mathematik\/ \bf201} (1959), 37--70.
```

Удобно использовать одну и ту же группу для `\sl` и для `\bf`. Здесь `\/` — это отделка, которую вы можете не понимать, пока до конца не прочтете главу 4.

4.2. `{\it Объясните ... печатать\/ {\rm прямое} слово ... предложения.}` Обратите внимание на положение курсивной поправки.

4.3. `\def\ic#1{\setbox0=\hbox{#1\/}\dimen0=\wd0 \setbox0=\hbox{#1}\advance\dimen0 by -\wd0}`.

4.4. Имена команд составляются из букв, а не из цифр.

4.5. Скажите `\def\sl{\it}` в начале и удалите другие определения `\sl`, которые могут присутствовать в вашем форматном файле (например, одно из них могло быть внутри макрокоманды `\tenpoint`).

4.6. `\font\squinttenrm=cmr10 at 5pt`
`\font\squinttenrm=cmr10 scaled 500`

5.1. `{shelf}ful`, `shelf{}ful`, и т.д., или даже `shelf\/ful`, которое дает shelfful вместо shelfful. В действительности, идея вставить курсивную поправку лучше, поскольку TEX будет повторно вставлять лигатуру ff после переноса `shelf{}ful`. (Приложение H отмечает, что лигатуры помещаются в переносимое слово, которое не содержит "явных кернов", а курсивная поправка является явным керном.) Но курсивная поправка может оказаться слишком большой (особенно в курсивном шрифте), и тогда `shelf{\kern0pt}ful` будет самым лучшим вариантом.

5.2. "␣{␣}␣" или "␣{}␣{}␣", и т.д. Начальный TEX также имеет макрокоманду `\space`, так что можно вводить `\space\space\space`. (Это не лучший эквивалент "\␣\␣\␣", так как пробелы регулируются при помощи "коэффициента пробела", как это объясняется позже.)

5.3. В первом случае вы получите такой же результат, как если бы самые внутренние скобки не появились совсем, поскольку группирование не использовалось для изменения шрифтов, команд или чего-нибудь еще, хотя TEX и не возражает, если вам хочется тратить время на создание групп без особенных причин. Но во втором случае были забыты необходимые скобки. Вы получите букву T в центре отдельной строки, за которой следует абзац, начинающийся с "ак должно быть.".

5.4. Вы получите тот же самый результат, как если бы вокруг "`\it centered`" была другая пара скобок, за тем исключением, что точка берется из курсивного шрифта. (Обе точки выглядят одинаково.) Шрифт `\it` не будет сохранятся после `\centerline`, но тут такое совпадение: TEX использует фигурные скобки, чтобы определить, какой текст центрируется, но затем убирает эти скобки. Операция `\centerline`, как она определена в приложении B, помещает результирующий текст без скобок в *другую* группу. Вот почему `\it` исчезает после `\centerline`. (Если вы еще не понимаете этого, то просто старайтесь не пропускать скобки в сомнительных ситуациях, и все будет о'кей.)

5.5. `\def\ital#1{{\it#1\/}}`. За: может показаться, что `\ital` легче выучить, поскольку она работает очень похоже на `\centerline` и не надо помнить о курсивной поправке. Против: избежав курсивной поправки непосредственно перед *запятой* или *точкой*, вероятно, надо будет выучить другую команду. Например, с помощью

```
\def\nocorr{\kern0pt }
```

пользователь может вводить "`\ital{comma} или \ital{period\nocorr},`". Альтернативный вариант, в котором нет курсивной поправки, а точки и запятые печатаются курсивом, выглядит не так уж хорошо. Длинная последовательность

курсива была бы для TEX'а неэффективной, поскольку текст для аргумента команды `\ital` должен целиком считываться в память только для того, чтобы снова сканироваться.

5.6. `{1 {2 3 4 5} 4 6} 4`.

5.7.
```
\def\beginthe#1{\begingroup\def\blockname{#1}}
\def\endthe#1{\def\test{#1}%
  \ifx\test\blockname\endgroup
  \else\errmessage{Вы должна сказать
    \string\endthe{\blockname}}\fi}
```

6.1. Лень и/или упрямство.

6.2. Здесь после "called —" появляется нежелательный пробел, поскольку, как говорит эта книга, TEX рассматривает конец строки как пробел. Этот пробел обычно вам и нужен, если только строка не кончается дефисом или тире, поэтому надо следить за строками с дефисами или тире на конце.

6.3. Он заменяет марашку, которая бросается в глаза в вашем выводе. (Эта марашка не появилась бы, если бы `\overfullrule` был задан равным `0pt` или если бы бокс был недозаполнен.)

6.4. Это пробел `\parfillskip`, который оканчивает абзац. В начальном TEX'е `\parfillskip` равен нулю, когда последняя строка абзаца является полной, следовательно, в выводе эксперимента 3 перед линейкой в действительности не появится никакого пробела. Но все вертикальные пропуски появляются в сообщении о переполнении бокса как пробелы, даже если они равны нулю.

6.5. Задайте `\hsize=1.5in \tolerance=10000 \raggedright \hbadness=-1`, а затем сделайте `\input story`. TEX сообщит плохость всех строк (за исключением последних строк абзацев, в которых заполнение клеем делает плохость нулевой.)

6.6.
```
\def\extraspace{\nobreak \hskip 0pt plus .15em\relax}
\def\dash{\unskip\extraspace---\extraspace}
```

(Если вы попробуете получить этот рассказ с размерами 2-дюйма и 1.5-дюйма, то заметите существенное улучшение. `\unskip` позволяет не ставить пробел перед `\dash`. TEX попытается сделать перенос перед `\dash`, а не перед `---` (ср. с приложением H). Команда `\relax` в конце `\extraspace` является предосторожностью на случай, когда следующим словом будет "`minus`".)

6.7. TEX удалит пять элементов: `1`, `i`, `n`, ␣, `\centerline`. (Пробел был в конце строки 2, а `\centerline` — в начале строки 3.)

6.8. Команда `\centerline` могла полностью определить команду `\ERROR` перед тем, как указать TEX'у посмотреть на `#1`. Поэтому TEX не интерпретирует команды, когда сканирует аргумент.

7.1. Были использованы три запрещенных символа. Должно быть напечатано

```
Procter \& Gamble's ... \$2, a 10\% gain.
```

(Сами факты тоже неверны.)

7.2. Обратные косые черты крайне редки в тексте, к тому же `\\` очень легко ввести нечаянно. Вот почему кажется, что лучше не резервировать `\\` для такого ограниченного использования. Наборщик может дать команде `\\` любое определение (включая `\backslash`).

7.3. 1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13. Активные символы (тип 13) — несколько особые объекты. В большинстве случаев они ведут себя как команды (например, когда вы говорите `\let\x=~` или `\ifx\x~`), но могут вести себя и как символьные элементы, когда появляются в списках элементов для `\uppercase` или `\lowercase` и когда стоят в нераскрытом виде после `\if` или `\ifcat`.

7.4. Ее окончит либо `>`, либо `}`, либо любой элемент категории 2. Затем уничтожается действие всех определений `\catcode` внутри группы, за исключением тех, которые были `\global`. У TEX'а нет никаких встроенных знаний о том, как сочетаются отдельные виды символов группирования. Новые номера категорий начинают действовать, как только переварено присваивание `\catcode`. Например,

```
{\catcode`\>=2 >
```

является законченной группой. Но без пробела после 2 она не была бы закончена, т.к. TEX прочел бы `>` и преобразовал его в элемент до того, как узнал, что был задан номер категорий. TEX всегда читает элемент, следующий за константой, перед тем, как оценивает эту константу. .

7.5. Если вы вводите `\message{\string~}` и `\message{\string\~}`, TEX отвечает, соответственно, `~` и `\~`. Чтобы получить $\backslash_{12}$ из `\string`, надо сделать обратную косую черту активным символом. Это можно сделать так:

```
{\catcode`/=0 \catcode`\\=13 /message{/string\}}
```

("Нулевая команда", которую вы получаете, когда между `\csname` и `\endcsname` нет элементов, не является решением этого упражнения, поскольку `\string` преобразует ее в `\csname\endcsname`. Есть, тем не менее, и другое решение: если параметр TEX'а `\escapechar`, который будет объяснен в одном из следующий "опасных" абзацев, отрицателен или больше 127, то `\string\\` работает.)

7.6. $\backslash_{12}$ $\mathtt{a}_{12}$ $\backslash_{12}$ $\sqcup_{10}$ $\mathtt{b}_{12}$.

7.7. `\def\ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax}`
Заметим, что такая команда должна использоваться с осторожностью. Она не может быть включена в условный текст, поскольку `\ifx` будет невидимым, пока `\ifundefined` не раскрыт.

7.8. Первый `\uppercase` дает `A\lowercase{BC}`, затем вы получаете `Abc`.

7.9. `\copyright\ \uppercase\expandafter{\romannumeral\year}`. (Это, безусловно, сложно; `\expandafter` раскрывает то, что стоит после `{`, а не после группы.)

7.10. (Мы предполагаем, что параметр `#2` — это не просто активный символ и что `\escapechar` находится между 0 и 127.)

```
\def\gobble#1{} % убирает один элемент
\def\appendroman#1#2#3{\edef#1{\def\noexpand#1{\csname
\expandafter\gobble\string#2\romannumeral#3\endcsname}}#1}
```

8.1. `%` трактовался бы как символ комментария, поскольку его номер категории равен 14. Таким образом, ни `%`, ни `}` не попали бы в глотку TEX'а, где обрабатываются числа. Когда у нас символы категорий 0, 5, 9, 14 или 15, должен быть использован дополнительный `\`. К тому же `\` не вредит, поэтому вы всегда можете использовать его для надежности.

8.2. (a) Оба символа прерывают текущую строку, но символ категории 5 мог быть преобразован в элемент $␣_{10}$ или `par`, в то время как символ категории 14 никогда не производит элемент. (b) Они производят символьные элементы с различными номерами категорий. Например, $\$_3$ — это не тот же самый элемент, что $\$_4$, поэтому пищеварительный процесс TEX'а будет трактовать их по-разному. (c) То же самое, что и (b), плюс тот факт, что имена команд трактуют буквы по-разному. (d) Нет. (e) Да; символы категории 10 в начале каждой строки игнорируются, поскольку каждая строка начинается в состоянии N. (f) Нет.

8.3. TEX только что прочел команду `\vship`, поэтому находился в состоянии S и был готов читать пробел перед "`1in`". Впоследствии, поскольку он был в состоянии S, он игнорировал этот пробел. Но если бы вы в ответ на сообщение об этой ошибке ввели `I\obeyspaces`, то увидели бы пробел. Между прочим, когда TEX печатает контекст сообщения об ошибке, нижняя пара строк идет из текстового файла, в то время, как другая пара строк — это списки элементов, которые читает TEX (если только они не начинаются с `<*>`: тогда это текст, вставленный при исправлении ошибки).

8.4. $\$_3$ x_{11} $\hat{}_7$ 2_{12} $\$_3$ $\sim_{13}$ $␣_{10}$ `TeX` $\hat{}\hat{}C_{12}$ $␣_{10}$. Последний пробел приходит от ⟨return⟩, помещенного в конце строки. Символьный код для `^^C` равен 3. Начальный пробел игнорируется, поскольку начало строки управляется состоянием N.

8.5. H_{11} i_{11} $!_{12}$ $␣_{10}$ `par` `par`. "␣" приходит от ⟨return⟩ в конце первой строки, вторая и третья строки вносят по `par`.

8.6. Два `^^B` не распознаются как последовательные символы верхнего индекса, т.к., первый `^^B` преобразуется в код 2, который не равен следующему символу `^`. Следовательно, результатом являются семь элементов: $\hat{}\hat{}B_7$ $\hat{}\hat{}B_7$ M_{11} `^^B` $␣_{10}$ $\hat{}\hat{}M_{11}$ `M^^M`. Последний из них — это команда, имя которой состоит из двух букв. Пробел после `\M` удаляется перед тем, как TEX вставляет элемент ⟨return⟩.

8.7. Оба варианта в тексте работают прекрасно. В частности, они сочетаются, как в `\lq\lq`, в лигатуры. Но определение в приложении B работает также с константами. Например, допустимы `\char\lq\%` и `\char\rq140`. (Конструкция же `\let\lq=‘` с константами не работала бы, поскольку кавычки в ⟨число⟩ должны быть из символьных элементов категории 12. После `\let\lq=‘` команда `\lq` не будет ни превращаться в элемент символа, ни сама *являться* таким символом).

9.1. `na\"\i ve`, `na{\"\i}ve` или `na\"{\i}ve`.

9.2. Belovèd protégé; rôle coördinator; soufflés, crêpes, pâtés, etc.

9.3. `\AE sop's \OE uvres en fran\c cais`.

9.4.
```
{\sl Commentarii Academi\ae\ scientiarum imperialis
petropolitan\ae\/} стали теперь  {\sl Akademi\t\i a  Nauk SSSR,
Doklady}.
```

9.5. `Ernesto Ces\'aro, P\'al Erd\H os, \O ystein Ore, Stanis\l aw \'Swier% czkowski, Serge\u\i\ \t Iur'ev, Mu\d hammad ibn M\^us\^a al-Khw\^arizm\^\i.`

9.6. Умляутом является `\H`, который недоступен в `\tt`, так что акцент акцент надо взять в другом шрифте. Например, `{\tt P\'al Erd{\bf\H{\tt o}}s}` использует жирный акцент, который в нужной степени темный.

9.7. `{\it Europe on {\sl\$}15.00 a day\/}`

9.8. Дополнительные скобки делают изменение шрифта локальным. С помощью аргумента акцент `\'` становится более совместимым с другими акцентами типа `\d`, которые получены из других символов без использования примитива `\accent`.

10.1. В точности 7227 pt.

10.2. −.013837 in, 0. mm, +42.1 dd, 3 in, 29 pc, 123456789 sp. (Строки текста в этом руководстве имеют ширину 29 pc.)

10.3. Первое не разрешено, поскольку восьмеричная запись не может быть использована с десятичной точкой. Однако второе законно, поскольку ⟨число⟩ может быть шестнадцатиричным в соответствии с правилом, упомянутым в главе 8. Оно означает 12 cc, что равно 144 dd ≈ 154.08124 pt. Третье также допустимо, так как ⟨десятичная строка⟩ может быть пустой. Это усложненный способ сказать 0 sp.

10.4.
```
\def\tick#1{\vrule height 0pt depth #1pt}
\def\\{\hbox to 1cm{\hfil\tick4\hfil\tick8}}
\vbox{\hrule\hbox{\tick8\\\\\\\\\\\\\\\\\\\\}}
```
(Можно также попробовать поместить отметки на каждом миллиметре, чтобы посмотреть, насколько хороша ваша система. Некоторые печатающие устройства не могут управлять сразу 101 линейкой.)

10.5. Например, скажите `\magnification=\magstep1 \input story \end`, для увеличения 1200, а `\magstep2` и `\magstep3` для 1440 и 1728. Необходимы три прогона, так как за один проход возможно не более одного увеличения. Результат может выглядеть забавно, если шрифты не существуют с указанными увеличениями.

10.6. Увеличение будет с коэффициентом 1.2. Поскольку шрифтом `\first` является `cmr10` at 12 pt, он после увеличения будет `cmr10` at 14.4 pt. Шрифтом `\second` будет `cmr10` at 12 pt. (TEX заменяет `12truept` на `10pt`, а окончательный вывод увеличивает его опять до 12 pt.)

11.1. Это `E` находится внутри бокса, который сам внутри бокса.

11.2. Идея в том, чтобы создать бокс и заглянуть внутрь. Например,

```
\setbox0=\hbox{\sl g\/} \showbox0
```

обнаруживает, что `\/` осуществлено помещением керна после символа. Дальнейший эксперимент показывает, что этот керн вставляется даже тогда, когда курсивная поправка равна нулю.

11.3. Высота, глубина и ширина охватывающего бокса должны быть достаточно большими, чтобы охватить все содержимое, так что результатом является

```
\hbox(8.98608+0.0)x24.44484
.\tenrm T
.\kern 1.66702
.\hbox(6.83331+0.0)x6.80557, shifted -2.15277
..\tenrm E
.\kern 1.25
.\tenrm X
```

(Вы, вероятно, предсказали высоту `8.9861`. Внутренние вычисления TEX'а проводятся в `sp`, а не в `pt`/100000, поэтому округление в пятом десятичном знаке не легко предсказать.)

11.4. В англоязычном наборе нет видимых применений таких симметричных боксов, и кажется бессмысленным вносить дополнительную общность в качестве бесполезного багажа, который, если даже и используется, то исключительно редко, единственно ради симметрии. Другими словами, в тот день, когда были рождены боксы, автор носил халат компьютерщика вместо мантии математика. Время покажет, было или нет это фундаментальной ошибкой!

11.5. Следующее решение основано на макрокоманде `\makeblankbox`, которая печатает границы бокса, используя линейки данной толщины снаружи и внутри этого бокса.Размеры бокса такие же, как `\box0`.

```
\def\dolist{\afterassignment\dodolist\let\next= }
\def\dodolist{\ifx\next\endlist \let\next\relax
  \else \\\let\next\dolist \fi
  \next}
\def\endlist{\endlist}
\def\hidehrule#1#2{\kern-#1%
  \hrule height#1 depth#2 \kern-#2 }
\def\hidevrule#1#2{\kern-#1{\dimen0=#1
    \advance\dimen0 by#2\vrule width\dimen0}\kern-#2 }
\def\makeblankbox#1#2{\hbox{\lower\dp0\vbox{\hidehrule{#1}{#2}%
    \kern-#1 % overlap the rules at the corners
    \hbox to \wd0{\hidevrule{#1}{#2}%
      \raise\ht0\vbox to #1{}% set the vrule height
      \lower\dp0\vtop to #1{}% set the vrule depth
      \hfil\hidevrule{#2}{#1}}%
    \kern-#1\hidehrule{#2}{#1}}}}
\def\maketypebox{\makeblankbox{0pt}{1pt}}
\def\makelightbox{\makeblankbox{.2pt}{.2pt}}
\def\\{\expandafter\if\space\next\
 \else \setbox0=\hbox{\next}\maketypebox\fi}
\def\demobox#1{\setbox0=\hbox{\dolist#1\endlist}%
  \copy0\kern-\wd0\makelightbox}
```

11.6.

```
\def\frac#1/#2{\leavevmode\kern.1em
\raise.5ex\hbox{\the\scriptfont0 #1}\kern-.1em
/\kern-.15em\lower.25ex\hbox{\the\scriptfont0 #2}}
```

12.1. $9 + 16$ единиц, $9 + 32$ единиц, $12 + 0$ единиц. (Но TEX считал бы такое большое растяжение "бесконечно плохим".)

12.2. "Что случится сейчас?" помещается на строке шириной `\hsize`, причем пробел слева в два раза больше пробела справа; "а сейчас?" помещается на следующей строке, прижатое к правому краю.

12.3. Первые две дают "переполненный бокс", если аргумент не помещается на строке, а третья вместо этого позволяет аргументу высовываться на поля. (Командой `\centerline` в начальном TEX'е является `\centerlinec`; эффект высовывания показывается в эксперименте с узкой колонкой в главе 6.) Если аргумент не содержит бесконечного клея, `\centerlinea` и `\centerlineb` действуют одинаково, но `\centerlineb` будет центрировать аргумент, который содержит клей "fil".

12.4. `Mr.~\& Mrs.~User were married by Rev.~Drofnats, who preached on Matt.~19\thinspace:\thinspace3--9.` (Такие тонкие пробелы традиционны для Библейских ссылок на главу и стих, но не предполагается, что вы это знаете. Начальный TEX определяет `\thinspace` как керн, а не как клей, следовательно, на тонком пробеле не будет разрыва между строками.)

12.5. `Donald~E.\ Knuth, ``Mathematical typography,'' {\sl Bull.\ Amer.\ Math.\ Soc.\ \bf1} (1979), 337--372.` (Но `\` после `E.` не необходимо из-за правила, которое вы выучите, если отважитесь на следующий опасный поворот.)

12.6. Есть несколько способов, но, вероятно, самый простой — "`\hbox{HACA}.`" или "`HACA\null.`". (Макрокоманда `\null` — это сокращение для `\hbox{}`.)

12.7. 1000, за следующими исключениями: 999 после `B`, `S`, `D`, и `J`; 1250 после запятой; 3000 после восклицательного знака, двойных правых кавычек и точек. Если бы точка была после `B` (т.е., если бы в тексте было сказано "`B. Sally`"), то коэффициент пробела после этой точки был бы равен 1000, а не 3000.

12.8. `\box3` имеет высоту 2 pt, глубину 4 pt, ширину 3 pt. Начиная от точки привязки `\box3` подвиньтесь вправо на .75 pt и вниз на 3 pt, и вы получите точку привязки `\box1`, или подвиньтесь вправо на 1 pt, чтобы получить точку привязки `\box2`.

12.9. Для того, чтобы гарантировать непрерывность, компоненты растяжимости и сжимаемости `\baselineskip` и `\lineskip` должны быть равны, а также `\lineskiplimit` должно быть равно нормальному промежутку `\lineskip`.

12.10. Да, но только потому, что ни один из его боксов не имеет отрицательной высоты или глубины. Он был бы в большей безопасности, если бы установил `\baselineskip=-1000pt`, `\lineskip=0pt` и `\lineskiplimit=16383pt`. (Макрокоманда начального TEX'а `\offinterlineskip` делает это.)

12.11. Междустрочный клей будет равен нулю, а натуральная высота $1+1-3+2 = 1$ pt (поскольку глубина `\box2` не включается в натуральную высоту), поэтому клей, когда установится, будет в конце концов `\vskip-1pt`. Таким образом, `\box3` имеет высоту 3 pt, глубину 2 pt, ширину 4 pt. Его точка привязки совпадает с точкой привязки `\box2`. Чтобы получить точку привязки `\box1`, надо подвинуться на 2 pt вверх и на 3 pt вправо.

12.12. Междустрочный клей будет равен 6 pt минус 3 fil. Окончательная глубина будет нулевой, потому что за `\box2` следует клей. Натуральная высота равна 12 pt, а сжимаемость — 5 fil. Поэтому `\box4` будет иметь высоту 4 pt, глубину 0 pt, ширину 1 pt, и будет содержать пять элементов: `\vskip-1.6pt`, `\box1`, `\vskip1.2pt`, `\moveleft4pt\box2`, `\vskip-1.6pt`. Начиная от точки привязки `\box4`, вы получите точку привязки `\box1`, поднявшись на 4.6 pt, или точку привязки `\box2`, подвинувшись на .4 pt вверх и на 4 pt влево. (Например, вы поднимаетесь наверх на 4 pt и получаете верхний левый угол `\box4`, затем вниз на -1.6 pt, т.е. вверх на 1.6 pt, чтобы получить верхний левый угол `\box1`, затем вниз на 1 pt, чтобы достичь его точки привязки. Эта проблема является чисто академической, поскольку довольно глупо включать в baselineskip бесконечную сжимаемость.)

12.13. Теперь `\box4` будет иметь высоту 4 pt, глубину -4 pt, ширину 1 pt, и в нем будут `\vskip-2.4pt`, `\box1`, `\vskip-1.2pt`, `\moveleft4pt\box2` и `\vskip-2.4pt`. От базовой линии `\box4` надо подняться на 5.4 pt, чтобы достичь базовой линии `\box1`, или на 3.6 pt, чтобы достичь базовой линии `\box2`.

12.14. `\vbox to` x{} дает высоту x, `\vtop to` x{} дает глубину x. Остальные размеры равны нулю. (Это справедливо даже когда x отрицательно.)

12.15. Есть несколько возможностей:

```
\def\nullbox#1#2#3{\vbox to#1{\vss\hrule height-#2depth#2width#3}}
```

работает, поскольку линейка будет нулевой толщины. Менее сложным является

```
\def\nullbox#1#2#3{\vbox to#1{\vss\vtop to#2{\vss\hbox to#3{}}}}.
```

Оба варианта действительны для отрицательной высоты и/или глубины, но не делают отрицательной ширины. Если отрицательной должна быть ширина, а не высота или глубина, можно использовать, например, такую конструкцию:

```
\def\nullbox#1#2#3{\hbox to#3{\hss\raise#1\null\lower#2\null}}.
```

Для `\hbox` невозможно создать бокс с отрицательной высотой или глубиной, а для `\vbox` или `\vtop` невозможно создать бокс с отрицательной шириной.

Однако, существует тривиальное решение этой проблемы, основанное на особенностях, которые мы будем обсуждать позднее:

```
\def\nullbox#1#2#3{\setbox0=\null
  \ht0=#1 \dp0=#2 \wd0=#3 \box0 }
```

12.16. `\def\llap#1{\hbox to 0pt{\hss#1}}`

12.17. Вы получите “Это” слева, а “загадка.” справа, поскольку пробел между словами имеет только растяжимость, которая конечна. Бесконечная растяжимость отбрасывается. (В этом случае правило TEX'a о бесконечном клее отличается от того , что мы получили бы в пределе, если бы значение 1 fil было конечным, но все более и более возрастающим. Самое предельное поведение таким же образом растянуло бы текст “Это загадка.”, но к тому же выдвинуло бы текст бесконечно далеко за правый край страницы.)

13.1. Если просто сказать `\hbox{...}`, то это не будет работать, поскольку бокс будет содержать только предыдущий вертикальный список без переключения мод. Вам надо явно начать абзац, и чтобы прямолинейным способом сделать это, надо сказать `\indent\hbox{...}`. Но предположим, что вы хотите определить макрокоманду, которая раскрывается в h-бокс, где эта макрокоманда используется как в середине абзаца, так и в его начале. Тогда вы не хотите заставлять пользователей ни вводить `\indent` перед вызовом вашей макрокоманды в начале абзаца, ни говорить `\indent` в самой макрокоманде (поскольку это могло бы вставить нежелательный отступ). Одно решение этой более общей проблемы — сказать `\␣\unskip\hbox{...}`, поскольку `\␣` делает моду горизонтальной, в то время как `\unskip` убирает нежелательный пробел. Начальный TEX предусматривает макрокоманду `\leavevmode`, которая решает эту проблему, вероятно, самым эффективным способом: `\leavevmode` является сокращением для `\unhbox\voidbox`, где `\voidbox` — это постоянно пустой регистр бокса.

13.2. Результат `\tracingcommands` показывает, что были переварены четыре элемента пробела. Они порождались на концах строк 2, 3, 4 и 5. Только первый из них имел какой-то эффект, поскольку в математических формулах и в вертикальных модах пробелы игнорируются.

13.3. "`end group character`" оканчивает абзац и `\vbox`, а `\bye` обозначает "`\vfill...`", поэтому следующими тремя строками являются

```
{math mode: math shift character $}
{restricted horizontal mode: end-group character }}
{vertical mode: \vfill}
```

13.4. Он содержит только смесь вертикального клея и горизонтальных линеек, точки привязки которых расположены слева на странице; никакого текста нет.

13.5. Вертикальная мода может встретиться только как самая внешняя мода. Горизонтальная и выделенная математическая моды могут появиться только когда они непосредственно охвачены вертикальной или внутренней вертикальной модой. Все другие случаи возможны.

14.1.
```
(см.~главу~12)
Главы 12 и~21
строка~16  из главы~6 файла {\tt story}
строки с 7 по~11
строки 2,~3, 4 и~5
(2)~большая черная полоса
Все 128~символов изначально имеют категорию~12
буква~{\tt x} в семействе~1
коэффициент~$f$, где $n$~в 1000~раз больше~$f$
```

14.2. "`Для всех $n$~больших, чем~$n_0$`", чтобы избежать отвлекающего разбиения.

14.3. "`упражнение \hbox{4.3.2--15}`" гарантирует, что после en-тире не будет разрыва. Но эта предосторожность редко необходима, так что "`exercise 4.3.2--15`" также допустимо. И не нужно никакой `~`; 4.3.2–15 такой длины, что не дает повода обижаться в начале строки.

14.4. Промежуток, который вы получаете из `~`, будет растягиваться и сжиматься вместе с другими промежутками той же самой строки, а промежутки внутри h-бокса имеют фиксированную ширину, поскольку клей в боксе уже установлен раз и навсегда. Кроме того, первый вариант допускает перенос слова "Глава".

14.5. `\hbox{$x=0$}` является неразрываемым и `${x=0}$`, как мы увидим позже, тоже не может быть разорвано. Оба эти режима устанавливают клей, окружающий знак равенства, в некоторое фиксированное значение. Но такому клею обычно хочется растянуться, к тому же решение с `\hbox` может включить нежелательный пробел в начале или конце строки, если `\mathsurround` не нуль. Третье решение `$x=\nobreak0$` свободно от обоих этих недостатков.

14.6. Штраф `\exhyphenpenalty=10000` запрещает все такие разрывы строк в соответствии с правилами, которые приводятся далее в этой главе. Аналогично, штраф `\hyphenpenalty=10000` предотвращает разрывы неявных (возможных) дефисов.

14.7. Во второй и четвертой строках сделан отступ при помощи дополнительного "квадрата" пробела, т.е., одного дополнительного em в текущем стиле печати. (Команда `\quad`, когда TEX находится в вертикальной моде, делает `\hskip`, который начинает новый абзац и помещает клей после отступа.) Если бы было использовано `\indent`, то эти строки не имели бы больший отступ, чем первая и третья, потому что `\indent` неявно имеется в начале каждого абзаца. Двойной отступ во второй и четвертой строках можно было получить при помощи `\indent\indent`.

14.8. `ba\ck en` и `Be\ttt uch`, где макрокоманды `\ck` и `\ttt` определены так:

```
\def\ck{\discretionary{k-}{k}{ck}}
\def\ttt{tt\discretionary{-}{t}{}}
```

Алгоритм переноса TEX'а автоматически такое изменение написания не делает.

14.9. `\def\break{\penalty-10000 }`

14.10. Вы получаете вынужденный разрыв, как если бы здесь не было `\nobreak`, поскольку `\break` не может быть отменено другим штрафом. Вообще, если вы имеете два штрафа подряд, их общий эффект такой же, как если бы был только один штраф, значение которого равно минимальному из первоначальных значений, если только оба эти значения не принуждают к разрывам. (Из `\break\break` получаются два разрыва; второй создает пустую строку.)

14.11. При $p \le -10000$ разрывы являются вынужденными, так что нет смысла в вычитании большой константы, эффект которого известен *заранее*, особенно если это может привести к арифметическому переполнению.

14.12. $(10+131)^2+0^2+10000 = 29881$ и $(10+1)^2+50^2+10000 = 12621$. В обоих случаях был добавлен `\adjdemerits`, поскольку строки были визуально несовместимыми (приличная, затем слишком свободная, потом опять приличная). Для `\linepenalty` и `\adjdemerits` были использованы значения начального TEX'а.

14.13. Потому, что в TEX'е отпадает клей, который появляется строго перед `\par`. Бен должен сказать, например, `\hfilneg\ \par`.

14.14. Просто скажите `\parfillskip=\parindent`. Но TEX не сможет найти подходящие разрывы строк, если абзац не достаточно длинный или не очень удачный. Но если текст хороший, результат будет безукоризненно симметричным.

14.15. Предполагая, что автор скончался и/или задал его или ее стиль, решением является вставить `{\parfillskip=0pt\par\parskip=0pt\noindent}` приблизительно после каждых 50 строк текста. (Каждый пробел между словами обычно является возможной точкой разрыва, если он находится достаточно далеко от начала абзаца.)

14.16. `{\leftskip=-1pt \rightskip=1pt` ⟨текст⟩ `\par}`

(Это применимо только к абзацу целиком. Если вы хотите исправить только отдельные строки, то должны это сделать вручную.)

14.17. `\def\line#1{\hbox to\hsize{\hskip\leftskip#1\hskip\rightskip}}` — и это все, что надо изменить. (Между прочим, выделенные уравнения не берут в расчет ни `\leftskip`, ни `\rightskip`. Изменить это намного труднее, поскольку может быть очень много вариаций.)

14.18. Наилучшее решение автора основано на переменной `\x` регистра `\dimen`:

```
\setbox1=\hbox{I}
\setbox0=\vbox{\parshape=11 -0\x0\x -1\x2\x -2\x4\x -3\x6\x
   -4\x8\x -5\x10\x -6\x12\x -7\x14\x -8\x16\x -9\x18\x -10\x20\x
  \ifdim \x>2em \rightskip=-\wd1
  \else \frenchspacing \rightskip=-\wd1 plus1pt minus1pt
   \leftskip=0pt plus 1pt minus1pt \fi
  \parfillskip=0pt \tolerance=1000 \noindent I turn, ... hand.}
\centerline{\hbox to \wd1{\box0\hss}}
```

Удовлетворительный результат при шрифте `cmr10` получается, когда `\x` устанавливается в 8.9 pt, 13.4 pt, 18.1 pt, 22.6 pt, 32.6 pt и 47.2 pt. При этом получаются треугольники, высота которых равна, соответственно, 11, 9, 8, 7, 6 и 5 строк.

14.19. `\item{}` в начале каждого абзаца, для которого желателен подвешенный отступ.

14.20. `\item{$\bullet$}`

14.21. Измените либо `\hsize`, либо `\rightskip`. Хитрость в том, чтобы вернуть их назад после окончания абзаца. Есть способ сделать это без группирования:

```
\let\endgraf=\par \edef\restorehsize{\hsize=\the\hsize}
\def\par{\endgraf \restorehsize \let\par=\endgraf}
\advance\hsize by-\parindent
```

14.22.
```
\dimen0=\hsize \advance\dimen0 by 2em
\parshape=3 0pt\hsize 0pt\hsize -2em\dimen0
```

14.23. Три абзаца могут быть составлены в один абзац, если вы после первых двух `\par` используете `\hfil\vadjust{\vskip\parskip}\break\indent`. Затем вы говорите, например, `\hangindent=-50pt \hangafter=-15`. (Та же самая идея может быть применена с `\looseness`, если вы хотите один из трех абзацев сделать

более свободным, но не хотите выбирать, какой. Однако, длинные абзацы заполняют память TEX'а, поэтому, пожалуйста, проявляйте сдержанность) Также посмотрите на следующее упражнение.

14.24. Используйте между абзацами макрокоманду `\hangcarryover`, определенную следующим образом:

```
\def\hangcarryover{\edef\next{\hangafter=\the\hangafter
    \hangindent=\the\hangindent}
  \par\next
  \edef\next{\prevgraf=\the\prevgraf}
  \indent\next}
```

14.25. Он будет размещать текущий абзац в минимальном числе строк, которое может быть достигнуто без нарушения допуска, и задав это число строк, он будет оптимально их разрывать. (Однако, ненулевая `\looseness` затрудняет работу TEX'а, так что это не рекомендуется, если вы не хотите платить за дополнительные вычисления. Можно получить почти тот же самый результат намного более эффективно, просто устанавливая `\linepenalty=100`.)

14.26. 150, 100, 0, 250. (Когда штраф равен нулю, как между строками 3 и 4, никакого штрафа в действительности не вставляется.)

14.27. `\interlinepenalty` плюс `\clubpenalty` плюс `\widowpenalty` (а также плюс `\brokenpenalty`, если первая строка заканчивается возможным разрывом).

14.28. Сложность в том, чтобы избежать "открытия" абзаца, добавляя что-то к его высоте. Кроме того, эта звездочка добавляется после строки, имеющей неизвестную глубину, поскольку глубина строки зависит от деталей разрыва строки, которые становятся известны позже. Следующее решение использует `\strut` и предполагает, что строка, содержащая на полях звездочку, имеет глубину, не превышающую `\dp\strutbox`, глубину `\strut`.

```
\def\strutdepth{\dp\strutbox}
\def\marginalstar{\strut\vadjust{\kern-\strutdepth\specialstar}}
```

Здесь `\specialstar` — это бокс нулевой высоты и глубины `\strutdepth`. Именно он помещает звездочку на левое поле:

```
\def\specialstar{\vtop to \strutdepth{
    \baselineskip\strutdepth
    \vss\llap{* }\null}}
```

14.29. `\def\insertbullets{\everypar={\llap{$\bullet$\enspace}}}`

(Аналогичное устройство может быть использовано, чтобы автоматически вставить подвешенный отступ и/или несколько абзацев.)

14.30. Сначала пойдет клей `\parskip` (но вы можете не увидеть его на текущей странице, если сказали `\showlists`, поскольку клей вверху каждой страницы исчезает). Затем идет результат `\everypar`, но давайте допустим, что `\everypar` ничего не добавляет к горизонтальному списку, так что вы получаете пустой горизонтальный список. Затем здесь отсутствует часть абзаца перед выделением. Выделенное уравнение подчиняется обычным правилам (оно занимает строки 1–3

абзаца, и использует отступ и длину строки 2, если форма нестандартная). За выделением не следует ничего, поскольку пробел после закрывающих `$$` игнорируется.

Между прочим, все будет иначе, если вы начнете абзац с `$$`, а не с `\noindent$$`, поскольку TEX вставит отступ абзаца, который появится на отдельной строке (с клеями `\leftskip`, `\parfillskip` и `\rightskip`).

14.31. Разрыв со штрафом `\penalty50` удалит `\hskip2em\nobreak\hfil`, так что следующая строка будет вынуждена начаться с имени рецензента, прижатого влево. (Но эффективнее использовать `\vadjust{}`, чем `\hbox{}`.)

14.32. Иначе алгоритм разбиения строк мог предпочесть две последние строки одной просто для того, чтобы передвинуть дефис из предпоследней строки в предпредпоследнюю, где это не приводит к увеличению дефектности. Это фактически привело бы к некоторым сюрпризам, когда тестировалась команда `\signed`. Для диагностики этой проблемы было использовано `\tracingparagraphs=1`.

14.33. Равномерное распределение дополнительных пробелов привело бы к трем строкам максимальной плохости (10000). Лучше иметь одну плохую строку вместо трех, поскольку TEX не различает степень плохости, когда строки действительно ужасны. В нашем частном случае `\tolerance` был 200, поэтому TEX не пытался сделать какое-либо разбиение строк, которое растянуло бы первые две строки, но даже если бы допуск был поднят до 10000, оптимальное расположение имело бы только одну незаполненную строку. Если вы действительно хотите равномерно расположить пробелы, то можете это сделать, используя `\spaceskip`, чтобы увеличить величину растяжимости между словами.

14.34.
```
\def\raggedcenter{\leftskip=0pt plus4em \rightskip=\leftskip
\parfillskip=0pt \spaceskip=.3333em \xspaceskip=.5em
\pretolerance=9999 \tolerance=9999
\hyphenpenalty=9999 \exhyphenpenalty=9999 }
```

15.1. Последние три строки вычислений для разбиения страниц были бы

```
% t=504.0 plus 5.0 minus 2.0 g=528.0 b=10000 p=150 c=100000#
% t=515.0 plus 5.0 minus 2.0 g=528.0 b=1755 p=-100 c=1655#
% t=543.0 plus 8.0 minus 4.0 g=528.0 b=* p=0 c=*
```

так что разрыв оказался бы на том же самом месте. Плохость была бы равна 1755, и страница уже не выглядела бы допустимой. (С другой стороны, если бы этот абзац был на две строки короче, чем он есть, то первые две строки следующего "опасного" абзаца появились бы на этой странице. Естественная высота $t = 532\,\mathrm{pt}$ имела бы возможность сжаться до $g = 528\,\mathrm{pt}$, поскольку два "medskip" на странице имели бы общую сжимаемость в 4 pt. Это, конечно, было бы предпочтительнее растянутой страницы, у которой плохость была 10000, но автор мог увидеть это и написать еще одно или два предложения, чтобы абзац не разрывался. Кроме всего, предполагается, что это руководство будет примером хорошей практической работы.)

15.2. Следующий разрешенный разрыв после опасного абзаца встречается на 28 pt позже, поскольку тут 6 pt дополнительного пробела для `\medskip`, за которыми следуют две строки по 11 pt каждая. TEX не позволяет разрыв между этими

двумя строками; `\clubpenalty` в приложении E установлено в 10000, поскольку обозначение опасного поворота имеет высоту в две строки.

15.3. Страница всегда содержит как минимум один бокс, если нет вставок, поскольку в противном случае законные точки разрыва отпадают. Утверждение (a) окажется неверным, если высота самого верхнего бокса превышает 10 pt, а утверждение (b) — если глубина самого нижнего бокса превышает 2.2 pt или если между самым нижним боксом и разбиением страницы окажется некоторый клей или керн (если только этот клей или керн явно не удаляют глубину бокса).

15.4. `\topinsert\vskip2in\rightline{\vbox{\hsize ... artwork.}}\endinsert` — вот что решает задачу. Но еще эффективней можно избежать `\rightline`, изменяя `\leftskip` следующим образом: `\leftskip=\hsize \advance\leftskip by-3in`. Тогда TEX не должен читать текст заголовка дважды.

15.5. Она появилась бы на странице 25, поскольку помещается там. `\midinsert` выскочит впереди других вставок, если она не переносится на следующую страницу. Например, если бы вторая трехдюймовая вставка была `\midinsert`, то она бы не появилась на странице 26, поскольку преобразовалась бы в `\topinsert`, т.к. макрокоманда `\midinsert` замечает, что вставка слишком велика для страницы 25.

15.6. Установите значения `\count1` в 50, `\dimen2` в 50 pt, `\count1` в 6, `\skip2` в −10 pt plus 6 fil minus 50 pt, `\skip2` в 60 pt plus −36 fil minus −300 pt, `\skip2` в 1 sp minus −6 sp, `\count6` в 1, `\skip1` в 25 pt plus 1 sp minus 1 fill, `\skip2` в 25 pt minus −150 pt, `\skip1` в 0 pt plus 1 sp minus 1 fill.

15.7. Если `\skip4` имеет бесконечную растяжимость, то `\skip5` будет нулем. Иначе он будет 0 pt plus 1 pt.

15.8.
```
\advance\dimen2 by0.5\dimen3 \divide\dimen2 by\dimen3
\multiply\dimen2 by\dimen3
```

15.9. `\count1` принимает значение 5, затем 2 (старое 5 спасается), затем 4 (которое делается глобальным), затем 8 (и 4 спасается). Наконец, значение 4 восстанавливается, и оно является ответом. (Для дальнейших замечаний см. обсуждение `\tracingrestores` в главе 27.)

15.10. `\hbox{\hbox{A}A}`. После `\unhbox5`, `\box5` является пустым; `\unhcopy5` не производит ничего.

15.11. `\hbox{A}`. Но после `{\global\setbox3=\hbox{A}\setbox3=\box3}`, `\box3` будет пустым.

15.12.
```
\newcount\notenumber
\def\clearnotenumber{\notenumber=0}
\def\note{\advance\notenumber by 1
  \footnote{$^{\the\notenumber}$}}
```

15.13. Да, при некоторых обстоятельствах. (1) Если нет другой законной точки разрыва, TEX выберет точку разрыва, стоимость которой равна ∞. (2) Если команда `\vadjust{\eject}` встречается на той же строчке, что и сноска, перед этой сноской, ссылка будет насильственно отделена. (3) Другие команды `\vadjust` на этой строке могли также вставить точки разрыва перед вставкой.

16.1. `$\gamma+\nu\in\Gamma$`.

16.2. `\le`, `\ge` и `\ne` (это сокращения для "меньше или равно", "больше или равно" и "не равно"). Также можно использовать имена `\leq`, `\geq` и `\neq`. (Четвертым наиболее общим символом является, вероятно, ∞, который обозначает "бесконечность" и называется `\infty`.)

16.3. В первой формуле `_2` применяется к знаку плюс ($x +_2 F_3$), а в последней — к пустой подформуле ($x + {}_2F_3$).

16.4. Результатами являются x^{y_z} и x^{y^z}. В первом случае z того же размера, что и y, а во втором она меньше. Более того, y и z в первом случае находятся на не совсем одинаковой высоте. (Хороший наборщик никогда даже и не подумает о первом случае, потому что математики его не любят.)

16.5. Вторая альтернатива работает неправильно, когда нижний индекс присутствует одновременно с `\prime`. Более того, некоторые математики используют `\prime` также в позиции нижнего индекса. Они, например, пишут $F'(w,z) = \partial F(w,z)/\partial z$ и $F_\prime(w,z) = \partial F(w,z)/\partial w$.

16.6. `$R_i{}^{jk}{}_l$`.

16.7. `10^{10}`; `2^{n+1}`; `(n+1)^2`; `\sqrt{1-x^2}`; `\overline{w+\overline z}`; `p_1^{e_1}`; `a_{b_{c_{d_e}}}`; `\root3\of{h''_n(\alpha x)}`. (Конечно, надо заключить эти формулы в знаки доллара, так что TEX будет обрабатывать их в математической моде. Верхние и нижние индексы могут быть заданы в любом порядке. Например, как `h''_n`, так и `h_n''` оба работают одинаково. Вы не должны убирать никаких скобок, показанных здесь; например, `$10^10$` дало бы 10^10. Но не повредит вставить дополнительные скобки вокруг букв или цифр, как в `({n}+{1})^{2}`. Указанные пробелы являются необходимыми, если только не используются дополнительные скобки, иначе TEX будет жаловаться на неопределенные команды `\overlinez` и `\alphax`.)

16.8. Он получил "Если$x = y\ldots$", поскольку забыл оставить пробел после "`Если`". Между знаками доллара пробелы исчезают. В конце предложения должно быть `$y$.`. Знак препинания, который принадлежит предложению, не включается в формулу, как мы увидим в главе 18. (Но еще не ожидается, что вы это знаете.)

16.9. `Если удалить элемент из $n$-множества, получится $(n-1)$-множество.`

16.10. Q, f, g, j, p, q, y. (Аналогичные греческие буквы $\beta, \gamma, \zeta, \eta, \mu, \xi, \rho, \phi, \varphi, \chi, \psi$.)

16.11. `$z^{*2}$` и `$h_*'(z)$`.

16.12. `$3{\cdot}1416$`. (Один из более ранних примеров в этой главе показал, что `\cdot` является бинарной операцией. Если же ее поместить в фигурных скобках, она действует как обычный символ.)

Если у вас много таких констант, как эта, например, в таблице, существует способ заставить обычные точки действовать как символы `\cdot`: определите просто `\mathcode‘.` равным `"0202` в предположении, что используются шрифты начального TEX'а. Однако это может быть опасным, поскольку в выделенных уравнениях часто используются обычные точки. Изменение `\mathcode` должно быть ограничено теми местами, в которых каждая точка является `\cdot`.

16.13. `$e^{-x^2}$`, `$D\sim p^\alpha M+l$` и `$\ghat\in(H^{\pi_1^{-1}})'$`. Если вы читали опасные секции, то знаете, что `\ghat` рекомендуется определять так: `\def\ghat{{\hat g}}`.

17.1. $x+y^{2/(k+1)}$ (`$x+y^{2/(k+1)}$`).

17.2. $((a+1)/(b+1))x$ (`$((a+1)/(b+1))x$`).

17.3. Он получил выделенную формулу

$$\frac{x=(y^2}{k+1)}$$

, потому что забыл, что несвязанный `\over` применяется ко всей подформуле. (Он, вероятно, должен был ввести `$$x=\left(y^2\over k+1\right)$$`, используя идеи, описанные в этой главе позже. Это не только увеличивает скобки, но и сохраняют "$x=$" вне дроби, поскольку `\left` и `\right` вводят подформулы.)

17.4. `$7{1\over2}\cents$` или `7$1\over2$\cents`. (Между прочим, здесь использовалось определение `\def\cents{\hbox{\rm\rlap/c}}`.)

17.5. Стиль D' используется для подформулы $p_2^{e'}$, следовательно, стиль S' используется для верхнего индекса e' и нижнего индекса 2, стиль SS' используется для штриха повторного индекса. Знак квадратного корня и p появляются в текстовом размере, 2 и e — в индексном размере, $\prime$ — в размере повторного индекса.

17.6. `$${1\over2}{n\choose k}$$`; `$$\displaystyle{n\choose k}\over2$$`. Все фигурные скобки необходимы.

17.7. `$${p \choose 2} x^2 y^{p-2} - {1 \over 1-x}{1 \over 1-x^2}.$$`

17.8. `$$\sum_{i=1}^p\sum_{j=1}^q\sum_{k=1}^ra_{ij}b_{jk}c_{ki}$$`.

17.9. `$$\sum_{{\scriptstyle 1\le i\le p \atop \scriptstyle 1\le j\le q} \atop \scriptstyle 1\le k\le r} a_{ij} b_{jk} c_{ki}$$`.

17.10. `$\displaystyle\biggl({\partial^2\over\partial x^2}+ {\partial^2\over\partial y^2}\biggr)\bigl|\varphi(x+iy)\bigr|^2=0$`.

17.11. Формулы, высота которых больше высоты одной строки, обычно имеют высоту две строки, а не высоту $1\frac{1}{2}$ или $2\frac{1}{2}$ строки.

17.12. `$\bigl(x+f(x)\bigr) \big/ \bigl(x-f(x)\bigr)$`. (Особо отметим `\big/` — обычная косая черта выглядела бы слишком маленькой перед `\big`-скобками.

17.13. `$$\pi(n)=\sum_{k=2}^n\left\lfloor\phi(k)\over k-1\right\rfloor.$$`

17.14. `$$\pi(n)=\sum_{m=2}^n\left\lfloor\biggl(\sum_{k=1}^{m-1}\bigl\lfloor(m/k)\big/\lceil m/k\rceil\bigr\rfloor\biggr)^{-1}\right\rfloor.$$`

17.15. Выделенная формула, эквивалентная следующей:

```
$${D}{{T}\over{T}^{{S}^{SS}}}$$.
```

17.16.
```
\def\sqr#1#2{{\vcenter{\vbox{\hrule height.#2pt
        \hbox{\vrule width.#2pt height#1pt \kern#1pt
           \vrule width.#2pt}
        \hrule height.#2pt}}}
\def\square{\mathchoice\sqr34\sqr34\sqr{2.1}3\sqr{1.5}3}
```

17.17. `\def\euler{\atopwithdelims<>}`.

17.18. Будет использован `\textfont0`, который был текущим в начале формулы, поскольку это переопределение является локальным в фигурных скобках. (Совсем другая история была бы, если бы вместо него появилось `\global\textfont`' — это изменило бы значение `\textfont0` на всех уровнях.)

17.19. `"2208` и `"220F`.

17.20. `\mathchardef\alpha="710B`. Между прочим, `{\rm\alpha}` будет давать неверный результат, поскольку символьная позиция `"0B` прямых шрифтов не содержит альфу. Вы должны предупредить ваших пользователей о том, какие символы им разрешено вводить под влиянием специальных соглашений типа `\rm`.

17.21. Если бы для `\delcode`{` был задан некоторый неотрицательный код ограничителя, то, когда вы печатали бы что-то вроде `\left{`, вы бы не получали сообщения об ошибке. Это плохо, потому что получался бы странный эффект, когда некоторые подформулы заданы в качестве аргументов макрокоманд или когда они появляются в выравниваниях. Но это имеет еще более серьезный недостаток, поскольку пользователь, который выиграет состязание с `\left{`, вероятно попробует также `\bigl{`, что кончится печально.

17.22. Поскольку `\bigl` определена как макрокоманда с одним параметром, она в качестве аргумента получает прямо `\delimiter`. Для того, чтобы выполнить эту работу, вы должны написать `\bigl{\delimiter"426830A}`. С другой стороны, `\left` будет мешать, если следующий символ является фигурной скобкой. Поэтому для всех ограничителей лучше иметь имена команд.

18.1. `$R(n,t)=O(t^{n/2})$, при $t\to0^+$.` (N.B.: `O(`, а не `O(`.)

18.2.
```
$$p_1(n)=\lim_{m\to\infty}\sum_{\nu=0}^\infty
  \bigl(1-\cos^{2m}(\nu!^n\pi/n)\bigr).$$
```

[Математики могут наслаждаться, интерпретируя эту формулу; см. G. H. Hardy, *Messenger of Mathematics* **35** (1906), 145–146.]

18.3.
```
\def\limsup{\mathop{\overline{\rm lim}}}
\def\liminf{\mathop{\underline{\rm lim}}}
```

[Заметим, что пределы $n \to \infty$ в обоих выделенных равенствах появляются на разных уровнях, поскольку "sup" и подчеркивание опускаются ниже базовой линии. Есть возможность поместить пределы на одном уровне, используя фантомы, что объяснено в этой главе позже. Например,

```
\def\limsup{\mathop{\vphantom{\underline{}}\overline{\rm lim}}}
```

дало бы нижние пределы в той же самой позиции, что и у `\liminf`.]

18.4. $x \equiv 0(\pmod y)^n$). Он должен был напечатать `$x\equiv0\pmod{y^n}$`.

18.5.
```
$${n\choose k}\equiv{\lfloor n/p\rfloor\choose
  \lfloor k/p\rfloor}{n\bmod p\choose k\bmod p}\pmod p.$$
```

18.6. `${\bf\bar x}^T\bf Mx={\rm0}\iff x=0$`. (Другим решением является `$\bf\bar x^{\mit T}Mx={\rm0}\iff x=0$`, но оно требует большего числа нажатий клавиш.)

18.7. `$S\subseteq{\mit\Sigma}\iff S\in{\cal S}$`. В этом случае фигурные скобки излишни и без них можно обойтись. Но не пытайтесь делать *все* наименьшим числом нажатий клавиш, или однажды вы перехитрите самого себя.

18.8.
```
$${\it available}+\sum_{i=1}^n\max\bigl({\it full}(i),
    {\it reserved}(i)\bigr)={\it capacity}.$$
```

[Если бы `\it` использовалась во всей формуле, нижний индекс i и верхний индекс n привели бы к такому сообщению об ошибке: `\scriptfont 4 is undefined`, поскольку в начальном TEX'е `\it` доступно только в текстовом размере.]

18.9.
```
{\obeylines \sfcode`;=3000
{\bf for $j:=2$ step $1$ until $n$ do}
\quad {\bf begin} ${\it accum}:=A[j]$; $k:=j-1$; $A[0]:=\it accum$;
\quad {\bf while $A[k]>\it accum$ do}
\qquad {\bf begin} $A[k+1]:=A[k]$; $k:=k-1$;
\qquad {\bf end};
\quad $A[k+1]:=\it accum$;
\quad {\bf end}.\par}
```

[Это нечто вроде "поэтического" примера в главе 14, но намного труднее. Некоторые руководства говорят, что знаки пунктуации должны быть того же шрифта, что и предшествующий им символ, так что должны быть напечатаны три вида точки с запятой: например, специалисты рекомендуют $k := j - 1$; $A[0] :=$ *accum;* **end;**. Автор с этим ни в коем случае не согласен.]

18.10.
```
Пусть $H$ --- Гильбертово пространство, \ $C$  --- замкнутое
ограниченное выпуклое  подмножество пространства~$H$, \
$T$ --- нерасширенное отображение~$C$.
Предположим, что при $n\to\infty$, \ $a_{n,k}\to0$ для  каждого~$k$, и
$\gamma_n=\sum_{k=0}^\infty(a_{n,k+1}-a_{n,k})^+\to0$.  Тогда для каждого
$x$~в~$C$,  \  $A_nx=\sum_{k=0}^\infty a_{n,k}T^kx$ слабо сходится
к фиксированной точке~$T$.
```

[Математик может либо оценить это, либо возмутиться, сделав попытку привести этот абзац к более приемлемому виду: "Пусть C — замкнутое, ограниченное, выпуклое подмножество Гильбертова пространства H, и пусть T — нерасширенное отображение C в себя. Предположим, что при $n \to \infty$, мы имеем $a_{n,k} \to 0$ для каждого k, и $\gamma_n = \sum_{k=0}^{\infty}(a_{n,k+1} - a_{n,k})^+ \to 0$. Тогда для каждого x in C, бесконечная сумма $A_n x = \sum_{k=0}^{\infty} a_{n,k} T^k x$ слабо сходится к фиксированной точке T."]

18.11.
```
$$\int_0^\infty{t-ib\over t^2+b^2}e^{iat}\,dt=
    e^{ab}E_1(ab),\qquad a,b>0.$$
```

18.12.
```
$$\hbar=1.0545\times10^{-27}\rm\,erg\,sec.$$
```

18.13. Здесь десять атомов (первым является f, а последним — y^2). Их типы и междуатомные пробелы, соответственно, таковы:

Ord Open Ord Punct \, Ord Close \; Rel \; Ord \> Bin \> Ord.

18.14.
```
$\left]-\infty,T\right[\times\left]-\infty,T\right[$.
```
(Или можно сказать `\mathopen` и `\mathclose` вместо `\left` и `\right`. Тогда TEX не выбирал бы

размеры ограничителей, а также считал бы, что подформулы имеют тип Inner.) Открытые интервалы более ясно выражаются, если использовать вместо перевернутых квадратных скобок круглые скобки. Например, сравните $(-\infty,T)\times(-\infty,T)$ с данной формулой.

18.15. Первый `+` будет атомом Bin, а второй — Ord. Следовательно результатом является x, средний пробел, $+$, средний пробел, $+$, нет пробела, 1.

18.16. `$x_1+x_1x_2+\cdots+x_1x_2\ldots x_n$` и `$(x_1,\ldots,x_n)\cdot(y_1,\ldots,y_n)=x_1y_1+\cdots+x_ny_n$`.

18.17. Запятые принадлежат предложению, а не формуле. Решение поместить их в математическую моду привело к тому, что TEX не сделал после них достаточные пробелы. К тому же его формулу $i = 1, 2, \ldots, n$ не разрешается нигде разрывать между строчками, кроме как после $=$, так что он рискует столкнуться с проблемами переполнения бокса. Но предположим, что предложение было более кратким:

Очевидно $a_i < b_i \quad (i = 1, 2, \ldots, n)$.

Тогда его идея была бы в основном правильной:

```
Очевидно $a_i<b_i$ \ ($i=1,2,\ldots,n$).
```

18.18. ... `никогда\footnote*{Ну \dots, почти никогда.} не должны` ...

18.19. Ни одна из этих формул не будет разорвана между строками, но толстые пробелы во второй формуле будут установлены в их натуральную ширину, в то время как толстые пробелы в первой формуле будут сохранять свою растяжимость.

18.20. Установить `\relpenalty=10000` и `\binoppenalty=10000`. Также понадобится изменить определения `\bmod` и `\pmod`, которые вставляют свои собственные штрафы.

18.21. `$\bigl\{\,x^3\bigm| h(x)\in\{-1,0,+1\}\,\bigr\}$`.

18.22. `$\{\,p\mid p$~and $p+2$ are prime$\,\}$`, предполагая, что значение `\mathsurround` равно нулю. Более трудная альтернатива `$\{\,p\mid p\ {\rm and}\ p+2\rm\ are\ prime\,\}$` не является решением, поскольку внутри математических формул разбиение строк не происходит на `\␣` (или клее любого вида). И конечно же, такую формулу лучше всего выделить, а не разрывать между строками.

18.23.
```
$$f(x)=\cases{1/3&if $0\le x\le1$;\cr 2/3&if $3\le x\le4$;\cr
0&elsewhere.\cr}$$
```

18.24.
```
$$\left\lgroup\matrix{a&b&c\cr d&e&f\cr}\right\rgroup
\left\lgroup\matrix{u&x\cr v&y\cr w&z\cr}\right\rgroup$$.
```

18.25. `\pmatrix{y_1\cr \vdots\cr y_k\cr}`.

18.26. `\def\undertext#1{$\underline{\smash{\hbox{#1}}}$}` будет подчеркивать слова и перечеркивать все опущенные ниже базовой линии символы. Или можно перед `\hbox` вставить `\vphantom{y}`, чтобы поместить все подчеркивания в позицию, которая ниже всех спущенных символов. Ни одна из этих возможностей не дает в точности то, что хочется. (См. также `\underbar` в приложении B.) Подчеркивание в книгопечатании в действительности не принято, поскольку, когда надо что-нибудь выделить, это лучше делать изменением шрифтов. Если вам действительно нужен подчеркнутый текст, лучше всего иметь специальный шрифт, в котором все буквы подчеркнуты.

18.27. `$n^{\rm ый}$ корень`. (Между прочим, в таких ситуациях также допустимо печатать "`$n$ый`", получая "*n*ый". Тот факт, что n печатается курсивом, отличает его от окончания. Рукописи, напечатанные на машинке, обычно изображают это с дефисом, но когда доступна курсивная n, "n-ый" не одобряется.)

18.28. `${\bf S^{\rm-1}TS=dg}(\omega_1,\ldots,\omega_n) =\bf\Lambda$`. (Вы заметили разницу между `\omega` (ω) и `w` (w)?)

18.29. `$\Pr(\,m=n\mid m+n=3\,)$`. (Аналогично множеству.)

18.30. `$\sin18^\circ={1\over4}(\sqrt5-1)$`.

18.31. `$k=1.38\times10^{-16}\rm\,erg/^\circ K$`.

18.32.
```
$\bar\Phi\subset NL_1^*/N=\bar L_1^*
  \subseteq\cdots\subseteq NL_n^*/N=\bar L_n^*$.
```

18.33. `$I(\lambda)=\int\!\!\int_Dg(x,y)e^{i\lambda h(x,y)}\,dx\,dy$`. (Хотя в выделенном стиле между последовательными знаками интеграла лучше срабатывают три `\!`, в текстовом стиле, по-видимому, достаточно только двух.)

18.34. `$\int_0^1\!\cdots\int_0^1f(x_1,\ldots,x_n)\,dx_1\ldots\,dx_n$`.

18.35.
```
$$x_{2m}\equiv\cases{Q(X_m^2-P_2W_m^2)-2S^2&($m$ нечетное)\cr
      \noalign{\vskip2pt} % spread the lines apart a little
      P_2^2(X_m^2-P_2W_m^2)-2S^2&($m$ четное)\cr}\pmod N.$$
```

18.36.
```
$$(1+x_1z+x_1^2z^2+\cdots\,)\ldots(1+x_nz+x_n^2z^2+\cdots\,)
  ={1\over(1-x_1z)\ldots(1-x_nz)}.$$
```
(Обратите внимание на использование `\,`.)

18.37.
```
$$\prod_{j\ge0}\biggl(\sum_{k\ge0}a_{jk}z^k\biggr)
  =\sum_{n\ge0}z^n\,\Biggl(\sum_
     {\scriptstyle k_0,k_1,\ldots\ge0\atop
      \scriptstyle k_0+k_1+\cdots=n}
   a_{0k_0}a_{1k_1}\ldots\,\Biggr).$$
```

Некоторые предпочли бы, чтобы последние скобки были крупнее, но в этом случае `\left` и `\right` получаются крупноватыми. Нетрудно определить макрокоманды `\bigggl` и `\bigggr` аналогичные определениям `\biggl` и `\biggr` в приложении B.

18.38.
```
$${(n_1+n_2+\cdots+n_m)!\over n_1!\,n_2!\ldots n_m!}
  ={n_1+n_2\choose n_2}{n_1+n_2+n_3\choose n_3}
    \ldots{n_1+n_2+\cdots+n_m\choose n_m}.$$
```

18.39.
```
$$\def\\#1#2{(1-q^{#1_#2+n})} % to save typing
\Pi_R{a_1,a_2,\ldots,a_M\atopwithdelims[]b_1,b_2,\ldots,b_N}
  =\prod_{n=0}^R{\\a1\\a2\ldots\\aM\over\\b1\\b2\ldots\\bN}.$$
```

18.40.
```
$$\sum_{p\rm\;prime}f(p)=\int_{t>1}f(t)\,d\pi(t).$$
```

18.41.
```
$$\{\underbrace{\overbrace{\mathstrut a,\ldots,a}
      ^{k\;a\mathchar''\rm s},
    \overbrace{\mathstrut b,\ldots,b}
      ^{l\;b\mathchar''\rm s}}_{k+l\rm\;элементов}\}.$$
```

Обратите внимание, как были получены апострофы (вместо прим).

18.42.
```
$$\pmatrix{\pmatrix{a&b\cr c&d\cr}&
              \pmatrix{e&f\cr g&h\cr}\cr
            \noalign{\smallskip}
            0&\pmatrix{i&j\cr k&l\cr}\cr}.$$
```

18.43.
```
$$\det\left|\,\matrix{
  c_0&c_1\hfill&c_2\hfill&\ldots&c_n\hfill\cr
  c_1&c_2\hfill&c_3\hfill&\ldots&c_{n+1}\hfill\cr
  c_2&c_3\hfill&c_4\hfill&\ldots&c_{n+2}\hfill\cr
  \,\vdots\hfill&\,\vdots\hfill&
       \,\vdots\hfill&&\,\vdots\hfill\cr
  c_n&c_{n+1}\hfill&c_{n+2}\hfill&\ldots&c_{2n}\hfill\cr
  }\right|>0.$$
```

18.44.
```
$$\mathop{{\sum}'}_{x\in A}f(x)\mathrel{\mathop=^{\rm def}}
  \sum_{\scriptstyle x\in A\atop\scriptstyle x\ne0}f(x).$$
```

Это работает, поскольку `{\sum}` имеет тип Ord (поэтому ее верхний индекс не располагается над ней), а `\mathop{{\sum}'}` имеет тип Op (поэтому нижний индекс располагается под ней). Однако пределы центрированы относительно $\sum'$, а не $\sum$. Если вам это не нравится, то средство будет более трудным: одно из решений — использовать `\sumprime_{x\in A}`, где `\sumprime` определено следующим образом:

```
\def\sumprime_#1{\setbox0=\hbox{$\scriptstyle{#1}$}
  \setbox2=\hbox{$\displaystyle{\sum}$}
  \setbox4=\hbox{${}'\mathsurround=0pt$}
  \dimen0=.5\wd0 \advance\dimen0 by-.5\wd2
  \ifdim\dimen0>0pt
    \ifdim\dimen0>\wd4 \kern\wd4 \else\kern\dimen0\fi\fi
  \mathop{{\sum}'}_{\kern-\wd4 #1}}
```

18.45.
```
$$2\uparrow\uparrow k\mathrel{\mathop=^{\rm def}}
  2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}
    \vbox{\hbox{$\Big\}\scriptstyle k$}\kern0pt}.$$
```

18.46. Если вам надо делать множество коммутативных диаграмм, следует определить некоторые такие макрокоманды, как те, которые находятся в первых нескольких строках этого решения. Макрокоманда `\matrix` переустанавливает базовые линии в `\normalbaselines`, поскольку команды типа `\openup` могли изменить

их. Поэтому в этом решении мы переопределяем `\normalbaselines`. Некоторое из того, что здесь показано, еще не объяснялось, но глава 22 раскроет все секреты.

```
$$\def\normalbaselines{\baselineskip20pt
  \lineskip3pt \lineskiplimit3pt }
\def\mapright#1{\smash{
    \mathop{\longrightarrow}\limits^{#1}}}
\def\mapdown#1{\Big\downarrow
  \rlap{$\vcenter{\hbox{$\scriptstyle#1$}}$}}
\matrix{&&&&&&0\cr
  &&&&&&\mapdown{}\cr
  0&\mapright{}&{\cal O}_C&\mapright\iota&
    \cal E&\mapright\rho&\cal L&\mapright{}&0\cr
  &&\Big\Vert&&\mapdown\phi&&\mapdown\psi\cr
  0&\mapright{}&{\cal O}_C&\mapright{}&
    \pi_*{\cal O}_D&\mapright\delta&
    R^1f_*{\cal O}_V(-D)&\mapright{}&0\cr
  &&&&&&\mapdown{\theta_i\otimes\gamma^{-1}}\cr
  &&&&&&\hidewidth R^1f_*\bigl({\cal O}
    _V(-iM)\bigr)\otimes\gamma^{-1}\hidewidth\cr
  &&&&&&\mapdown{}\cr
  &&&&&&0\cr}$$
```

19.1. `$$\sum_{n=0}^\infty a_nz^n\qquad\hbox{сходится, если}\qquad |z|<\Bigl(\limsup_{n\to\infty}\root n\!\of{|a_n|}\,\Bigr)^{-1}.$$`

```
$${f(x+\Delta x)-f(x)\over\Delta x}\to f'(x)
    \qquad\hbox{as $\Delta x\to0$.}$$

$$\|u_i\|=1,\qquad u_i\cdot u_j=0\quad\hbox{если $i\ne j$.}$$

$$\it\hbox{Конфлуентный образ}\;\left\{
    \matrix{\hbox{дуги}\hfill\cr\hbox{окружности}\hfill\cr
      \hbox{a fan}\hfill\cr}
    \right\}\;\hbox{есть}\;\left\{
    \matrix{\hbox{дуга}\hfill\cr
      \hbox{дуга $\backslash$ окружность}\hfill\cr
      \hbox{сектор или дуга}\hfill\cr}\right\}.$$
```

Первый пример включает '`\!`' и '`\,`', чтобы дать более изысканное распределение пробелов, но суть проблемы была в том, чтобы проиллюстрировать h-бокс, а не суетиться по поводу таких дополнительных подробностей. Последний пример может быть сделан намного проще, используя идеи главы 22, если вы не собираетесь опускаться на уровень примитивов TEX'а. Первую матрицу можно было бы заменить на

```
\,\vcenter{\halign{#\hfil\cr дуга\cr окружность a fan\cr}}\,
```

и вторую — аналогично.

19.2. `$$\textstyle y={1\over2}x$$`. (В многострочных формулах часто применяется переключение в текстовый стиль. Например, вы, вероятно, найдете случай использовать `\textstyle` с обеих сторон от `&` внутри `\eqalign`.)

19.3. Последняя формула будет в текстовом, а не в выделенном стиле. И даже если вы вводите `$$\hbox{$\displaystyle{`⟨формула⟩`}$}$$`, результаты будут не одинаковы, как мы это увидим позже: TEX будет сжимать клей в `$$`⟨формула⟩`$$`, если эта формула слишком широка, чтобы поместиться на строке со своей натуральной шириной, а внутри `\hbox{...}` клей "заморожен" со своей натуральной шириной.

19.4. Одно решение — это поместить формулу в h-бокс, который заполняет целую строку:

```
$$\leftline{\indent$\displaystyle
    1-{1\over2}+{1\over3}-{1\over4}+\cdots=\ln2$}$$
```

Но это увеличивает объем работы. Если вы сделаете определения

```
\def\leftdisplay#1$${\leftline{\indent$\displaystyle{#1}$}$$}
\everydisplay{\leftdisplay}
```

то можете вводить, как обычно, `$$`⟨формула⟩`$$`, и форматирование будет выполнено автоматически. (Этот способ не работает с номерами уравнения. Приложение D иллюстрирует, как справиться с ними.)

19.5.
```
$$\prod_{k\ge0}{1\over(1-q^kz)}=
  \sum_{n\ge0}z^n\bigg/\!\!\prod_{1\le k\le n}(1-q^k).\eqno(16')$$
```

19.6. `\eqno\hbox{(3--1)}`.

19.7. Когда вы вводите звездочку в математической моде, начальный TEX считает `*` бинарной операцией. В случае `(*)` и `(**)` бинарные операции преобразуются в тип Ord, поскольку они не появляются в бинарном контексте; но средние звездочки в `(***)` сохраняют тип Bin. Так что результат был $(***)$. Чтобы избежать дополнительных средних пробелов, можно вводить `\eqno(*{*}*)`, или, если вы никогда не используете `*` как бинарную операцию, изменить `\mathcode'*`.

19.8. Предполагая, что `\hsize` меньше, чем 10000 pt, натуральная величина этого уравнения будет слишком велика, чтобы помещаться на строке, к тому же `\quad` задает клей слева. Поэтому, $x = y$ появится в точности на расстоянии 1 em от левого края, а (5) окажется прижатым вплотную к правому краю. (Ширины будут удовлетворять $w = z - q$, $d = 0$, $k = q - e = 18$ mu.) В случае `\leqno`, (5) появится вплотную к левому краю, за ним один квадрат пробела в `\textfont2`, далее один квадрат пробела в текущем текстовом шрифте, за ним $x = y$.

19.9. (Заметим, в частности, что последняя "`.`" идет *перед* последним `\cr`.)

```
$$\eqalign{T(n)\le T(2^{\lceil\lg n\rceil})
    &\le c(3^{\lceil\lg n\rceil}-2^{\lceil\lg n\rceil})\cr
    &<3c\cdot3^{\lg n}\cr
    &=3c\,n^{\lg3}.\cr}$$
```

19.10.
```
$$\eqalign{P(x)&=a_0+a_1x+a_2x^2+\cdots+a_nx^n,\cr
   P(-x)&=a_0-a_1x+a_2x^2-\cdots+(-1)^na_nx^n.\cr}\eqno(30)$$
```

19.11. Обе части этого уравнения будут рассматриваться как левые, поэтому вы получите результат, который выглядит так:

$$\left\{\begin{aligned} \alpha &= f(z) \\ \beta &= f(z^2) \\ \gamma &= f(z^3) \end{aligned}\right\}.$$

19.12.
```
$$\leqalignno{\gcd(u,v)&=\gcd(v,u);&(9)\cr
    \gcd(u,v)&=\gcd(-u,v).&(10)\cr}$$
```

19.13.
```
$$\eqalignno{\biggl(\int_{-\infty}^\infty e^{-x^2}\,dx\biggr)^2
  &=\int_{-\infty}^\infty\int_{-\infty}^\infty
    e^{-(x^2+y^2)}\,dx\,dy\cr
  &=\int_0^{2\pi}\int_0^\infty e^{-r^2}r\,dr\,d\theta\cr
  &=\int_0^{2\pi}\biggl(-{e^{-r^2}\over2}
    \bigg|_{r=0}^{r=\infty}\,\biggr)\,d\theta\cr
  &=\pi.&(11)\cr}$$
```

19.14. Получится выделенный бокс

$$x = y + z$$

и

$$x^2 = y^2 + z^2.$$

Причина: ‘ “и” встречается в левой части бокса `\eqalign`, а не целого выделения, и бокс `\eqalign` центрируется как обычно.

19.15. Поднимая номер уравнения, он увеличил высоту строки, поэтому TEX, когда вычислил междустрочечный клей, поместил между этой и предыдущей строками дополнительный пробел. Если бы он сказал `\smash{\raise...}`, то не имел бы такой проблемы.

19.16.
```
$$\displaylines{\hfill x\equiv x;\hfill\llap{(1)}\cr
    \hfill\hbox{if}\quad x\equiv y\quad\hbox{then}\quad
       y\equiv x;\hfill\llap{(2)}\cr
    \hfill\hbox{if}\quad x\equiv y\quad\hbox{and}\quad
       y\equiv z\quad\hbox{then}\quad
       x\equiv z.\hfill\llap{(3)}\cr}$$
```

Существует также более хитрое решение, которое начинается с

```
$$\displaylines{x\equiv x;\hfil\llap{(1)}\hfilneg\cr
```

19.17.
```
$$\eqalignno{x_nu_1+\cdots+x_{n+t-1}u_t
    &=x_nu_1+(ax_n+c)u_2+\cdots\cr
    &\qquad+\bigl(a^{t-1}x_n+c(a^{t-2}+\cdots+1)\bigr)u_t\cr
    &=(u_1+au_2+\cdots+a^{t-1}u_t)x_n+h(u_1,\ldots,u_t).
      \quad&(47)\cr}$$
```

Никто не ожидает, что вы сразу вставите `\quad` на последней строке. Такая отделка обычно не предполагается, пока вы не увидите первые результаты. Но без этого `\quad` (47) окажется на пол квадрата ближе к формуле.

19.18.
```
$$\displaylines{\quad\sum_{1\le j\le n}{1\over
    (x_j-x_1)\ldots(x_j-x_{j-1})(x-x_j)(x_j-x_{j+1})
    \ldots(x_j-x_n)}\hfill\cr
  \hfill={1\over(x-x_1)\ldots(x-x_n)}.\quad(27)\cr}$$
```

19.19.
```
$$\def\\#1;{(#1;q^2)_\infty} % для сохранения печати
\displaystyle{q^{{1\over2}n(n+1)}\\ea;\\eq/a;\qquad\atop
  \hfill\\caq/e;\\cq^2\!/ae;}
\over(e;q)_\infty(cq/e;q)_\infty$$
```

20.1.
```
\def\mustnt{Я не должен болтать в классе.\par}
\def\five{\mustnt\mustnt\mustnt\mustnt\mustnt}
\def\twenty{\five\five\five\five}
\def\punishment{\twenty\twenty\twenty\twenty\twenty}
```

Решение более сложных задач этого типа обсуждаются позднее.

20.2. ABCAB. (Первое `\a` раскрывается в `A\def\a{B...}`; это переопределяет `\a`, так что второе `\a` раскрывается в `B...`, и т.д.) Именно это происходит, если TEX, когда строит список, встречает `\puzzle`. Но если `\puzzle` раскрывается в `\edef`, `\message` или что-нибудь подобное, мы позже увидим, что внутренние команды `\def` не выполняются в то время, когда имеет место раскрытие, поэтому результатом является бесконечная строка

```
A\def\a{B\def\a{C\def\a{A\def\a{B\def\a{C\def\a{A...
```

которая приводит к прерыванию TEX'a, поскольку входной стэк команды конечен. Этот пример указывает на то, что команду (например, `\b`) не надо определять, когда она появляется в тексте замены определения. Пример также показывает, что TEX не раскрывает макрокоманду до тех пор, пока it needs to.

20.3. ($\mathbf{x_1}, \ldots, \mathbf{x_n}$). Заметим, что индексы здесь введены жирным шрифтом, поскольку раскрытие `($\bf x_1,\ldots,\bf x_n$)` не "выключает" `\bf`. Чтобы предотвратить это, вы должны писать `\row{{\bf x}}` или (лучше) `\row\xbold`, вместе с `\def\xbold{{\bf x}}`.

20.4. Дело в том, что параметры попадут в макрокоманду `\mustnt`, если вы расширите предыдущий ответ:

```
\def\mustnt#1#2{Я не должен #1 в #2.\par}
\def\five#1#2{\mustnt{#1}{#2}...\mustnt{#1}{#2}}
\def\twenty#1#2{\five{#1}{#2}...\five{#1}{#2}}
\def\punishment#1#2{\twenty{#1}{#2}...\twenty{#1}{#2}}
```

Когда вы этим способом передаете параметры из одной макрокоманды в другую, вы должны заключить их в скобки, что и показано. Но в действительности это частное решение наказывает TEX намного больше, чем надо, поскольку он тратит много времени на копирование параметров и читает их снова и снова. Можно намного эффективнее выполнить эту работу, определив команды:

```
\def\mustnt{Я не должен \doit\ в \thatplace.\par}
\def\punishment#1#2{\def\doit{#1}\def\thatplace{#2}%
  \twenty\twenty\twenty\twenty\twenty}
```

и определив `\five` и `\twenty` без параметров, как и прежде. Можно глубже покопаться в TEXнических возможностях, чтобы найти решение, которое еще более эффективно. TEX работает быстрее, когда макрокоманды связаны друг с другом через боксы. Например,

```
\def\mustnt{\copy0 }
\def\punishment#1#2{\setbox0=
  \vbox{\strut Я не должен #1 в #2.\strut}%
  \twenty\twenty\twenty\twenty\twenty}
```

с большой скоростью задает 100 идентичных абзацев, поскольку TEX должен обработать абзац и разбить его на строки только один раз. Намного быстрее копировать бокс,чем строить его. (Подпорки в этом примере обеспечивают правильное расстояние между базовыми линиями боксированных абзацев, как объяснялось в главе 12. Две подпорки использованы для того, чтобы, если послание занимает более одной строки, подпорки были бы и в верхней строке, и в нижней. Если известно, что каждое предложение будет занимать только одну строку, то никакие подпорки не нужны, поскольку, когда бокс, созданный командой `\copy`, добавляется к текущему вертикальному списку, обычно добавляется междустрочный клей.)

20.5. Знаки `##` обязательны, когда текст замены определения содержит другие определения. Например, рассмотрим

```
\def\a#1{\def\b##1{##1#1}}
```

после которого `\a!` будет раскрываться в `\def\b#1{#1!}`. Позднее мы увидим, что `##` также важно и для выравниваний, смотри например, определение `\matrix` в приложении B.

20.6. `\def\a#{\b}`.

20.7. Давайте не спеша разберемся с этим, так чтобы ответ дал достаточно информации, чтобы отвечать на все подобные вопросы. Текст параметров определения состоит из трех элементов `#1`, `#2`, $[_1$, а текст замены состоит из шести элементов $\{_1$, $\#_6$, $]_2$, $!_6$, `#2`, $[_1$. (Когда в тексте замены встречаются два символа категории 6, остается второй; первый символ категории 6 не учитывается. Таким образом, `\def\!#1!2#[{##]!!#2]` произвело бы, по существу, идентичное определение.) Когда раскрывается данный список элементов, аргумент `#1` равен x_{11}, поскольку он неограничен. Аргумент `#2` ограничен $[_1$, xnj отличается от $\{_1$, поэтому он временно устанавливается в `{[y]]`; но внешние "скобки" снимаются, так что `#2` сокращается до трех элементов $[_1$, y_{11}, $]_2$. Поэтому результатом раскрытия является

$$\{_1\ \#_6\ ]_2\ !_6\ [_1\ y_{11}\ ]_2\ [_1\ z_{11}\ \}_2.$$

Между прочим, если вы поставите `\tracingmacros=1`, TEX скажет

```
\!!1#2[->{##]!!#2[
#1<-x
#2<-[y]
```

Номера категорий не показаны, но символы категории 6 всегда появляются дважды подряд. Параметрический элемент в тексте замены использует символьный код последнего параметра в параметрическом тексте параметра.

20.8. Да, конечно. В первом случае `\a` получает значение `\b`, которое является текущим в момент `\let`. Во втором случае `\a` становится макрокомандой, которая раскрывается в `\b`, только когда используется, так что она имеет то значение `\b`, которое является текущим в момент использования `\a`. Если вы хотите обменять значения `\a` и `\b`, вам нужна `\let`.

20.9. (a) Да. (в) Нет; может появиться другая команда (за исключением тех, которые объявлены как `\outer`-макрокоманды).

20.10.
```
\def\overpaid{{\count0=\count\balance
  Вы внесли сверх налога \dollaramount.
  \ifnum\count0<100 Мы возмещаем
    такие маленькие суммы только по вашей просьбе.
  \else Чек на эту сумму будет послан
    в отдельном конверте.\fi}}
```

20.11. Хитрость в том, чтобы получить нуль в числе `$2.01`.

```
\def\dollaramount{\count2=\count0 \divide\count2 by100
  \$\number\count2.%
  \multiply\count2 by-100 \advance\count2 by\count0
  \ifnum \count2<10 0\fi
  \number\count2 }
```

20.12.
```
\def\category#1{\ifcase\catcode`#1
  escape\or начало группы\or конец группы\or математическое\or
  выравнивание\or конец строки\or параметр\or верхний индекс\or
  нижний индекс\or игнорируемые\or пробел\or буква\or
  другой символ\or активный\or комментарий\or ошибочный\fi}
```

20.13. (a,b) Истинно. (c,d) Ложно. (e,f) Истинно. В случае (e), ⟨истинный текст⟩ начинается с "`ue`". (g) `\ifx` ложно, а внутренний `\if` истинный, поэтому внешний `\if` становится "`\if True...`", что ложно. (Интересно, TEX знает, что внешний `\if` ложный даже перед тем, как посмотреть на `\fi`, которые закрывают `\ifx` и внутренний `\if`.)

20.14. Одна идея — это сказать

```
\let\save=\c \let\c=0 \edef\a{\b\c\d} \let\c=\save
```

поскольку команды эквивалентны символам, которые не раскрываются. Однако, это не раскрывает те `\c`, которые могли присутствовать в раскрытиях `\b` и `\d`. Другой способ, который свободен от этого недостатка, это

```
\edef\next#1#2{\def#1{\b#2\d}} \next\a\c
```

(и он заслуживает самого внимательного отношения!).

20.15.
```
\toks0={\c} \toks2=\expandafter{\d}
\edef\a{\b\the\toks0 \the\toks2 }
```

(Заметим, что здесь `\expandafter` раскрывает элемент после левой фигурной скобки.)

20.16. Это решение не следует принимать слишком серьезно, хотя оно и работает:

```
{\setbox0=\vbox{\halign{#{\c\span\d}\cr
      \let\next=0\edef\next#1{\gdef\next{\b#1}}\next\cr}}}
\let\a=\next
```

20.17. Ни одно, хотя `\a` и будет, когда раскроется, вести себя как непарная левая фигурная скобка. Определение `\b` *не полное*, поскольку оно раскрывается в `\def\b{{}`. TEX будет продолжать читать вперед, ища вторую правую скобку, возможно обнаруживая сбежавшее определение! Невозможно определить макрокоманду, которая имеет непарные скобки. Но вы *можете* сказать `\let\a={`; приложение D обсуждает некоторые другие фокусы с фигурными скобками.

20.18. Один способ — переопределить `\catcode‘\^^M=9` (игнорируемый) сразу после `\read`, так что ⟨return⟩ будет игнорироваться. Другое решение — переопределить `\endlinechar=-1`, так что в конце строки не помещается никакого символа. Или можно попробовать схитрить следующим образом:

```
\def\stripspace#1 \next{#1}
\edef\myname{\expandafter\stripspace\myname\next}
```

Последнее решение не работает, если пользователь вводит `%` в конце своего имени или если имя содержит команды.

20.19. Есть два решения:

```
\def\next#1\endname{\uppercase{\def\MYNAME{#1}}}
\expandafter\next\myname\endname
```

```
\edef\next{\def\noexpand\MYNAME{\myname}}
\uppercase\expandafter{\next}
```

20.20. (Это решение к тому нумерует строки, поэтому число повторений можно легко изменить. Единственная хитрость здесь — это использование `\endgraf`, которая является заменой для `\par`, поскольку `\loop` — не `\long`-макрокоманда.)

```
\newcount\n
\def\punishment#1#2{\n=0
  \loop\ifnum\n<#2 \advance\n by1
    \item{\number\n.}#1\endgraf\repeat}
```

21.1. Междустрочный клей добавляется к v-боксам, но не добавляется к линейкам. Он забыл сказать `\nointerlineskip` перед и после конструкции `\moveright`.

21.2. `\vrule height3pt depth-2pt width1in`. Заметим, что было необходимо задать `\vrule`, поскольку линейка появилась в горизонтальной моде.

21.3. `\def\boxit#1{\vbox{\hrule\hbox{\vrule\kern3pt \vbox{\kern3pt#1\kern3pt}\kern3pt\vrule}\hrule}}`

(Результирующий бокс имеет не ту базовую линию, что исходный; чтобы получить ту, вы должны еще немного потрудиться .)

21.4. `\leaders`: два бокса, начинающиеся с 100 pt и 110 pt.
`\cleaders`: три бокса, начинающиеся с 95 pt, 105 pt и 115 pt.
`\xleaders`: три бокса, начинающиеся с 93 pt, 105 pt и 117 pt.

21.5.
```
\def\leaderfill{\kern-0.3em\leaders\hbox to 1em{\hss.\hss}%
    \hskip0.6em plus1fill \kern-0.3em }
```

21.6. Поскольку за `\vrule` не следует ни `height`, ни `depth`, высота и глубина будут '`*`'; т.е., линейка расширяется до наименьшего охватывающего бокса. Это обычно создает мрачную черную ленту, которая слишком ужасна, чтобы быть здесь показанной. Однако, это работает в макрокоманде `\downbracefill` приложения B; а `\leaders\vrule\vfill` прекрасно работает в вертикальной моде.

21.7. Например, скажите

```
\null\nobreak\leaders\hrule\hskip10pt plus1filll\ \par
```

'`\␣`' обеспечивает дополнительный клей, который уничтожается `\unskip`, подразумеваемым в конце каждого абзаца (см. главу 14), а '`\null\nobreak`' создает уверенность, что проводники не исчезнут при разбиении строки; '`filll`' погашает клей `\parfillskip`.

21.8.
```
$$\hbox to 2.5in{\cleaders
    \vbox to .5in{\cleaders\hbox{\TeX}\vfil}\hfil}$$
```

21.9. Мы предполагаем, что подпорка имеет высоту 12 pt и что на странице помещается 50 строк:

```
\setbox0=\hbox{\strut Я не должен болтать в классе.}
\null\cleaders\copy0\vskip600pt\vfill\eject % 50 раз на странице 1;
\null\cleaders\box0\vskip600pt\bye % еще 50 на странице 2.
```

`\null` предохраняет клей (и проводники) от исчезновения наверху страницы.

21.10. `{\let\the=0\edef\next{\write\cont{`⟨список элементов⟩`}}\next}` будет раскрывать все, кроме `\the`, когда встречается команда `\write`.

22.1. Заметим, что `\smallskip` используется здесь, чтобы отделить заголовок и подножие таблицы от самой таблицы; такая отделка часто имеет смысл.

```
\settabs\+\indent&10\frac1/2 фунтов\qquad&\it Порции\qquad&\cr
\+&\negthinspace\it Вес&\it Порции&
  {\it Примерное время приготовления\/}*\cr
\smallskip
\+&8 фунтов&6&1 час 50 -- 55 минут\cr
\+&9 фунтов&7 -- 8&Около 2 часов\cr
\+&9\frac1/2 фунтов&8 -- 9&2 часа 10 -- 15 минут\cr
\+&10\frac1/2 фунтов&9 to 10&2 часа 15 -- 20 минут\cr
\smallskip
\+&* Для фаршированного гуся добавьте   от 20 до 40 минут.\cr
```

В строке заголовка `\it` указано три раза, поскольку каждый элемент между позициями табуляции считается группой. Вы получите массу сообщений об ошибках, если попытаетесь сказать что-нибудь вроде `\+&{\it Вес&Порции&...}\cr`. Команда `\negthinspace` в строке заголовка — это маленький обратный пробел, который компенсирует наклон в курсивном W; автор вставил эту несколько необычную коррекцию после того, как увидел, как таблица выглядит без этого при первой попытке. (Не предполагается, что вы думаете о таких вещах, но это следует упомянуть.) Посмотрите упражнение 11.6 для макрокоманды `\frac`: в кулинарной книге лучше напечатать "$^1\!/_2$", чем "$\frac12$".

Другой способ напечатать эту таблицу — это поместить ее в вертикальный бокс вместо того, чтобы включать первую колонку, единственное назначение которой — это задать отступ.

22.2. В таких программах кажется лучше всего писать `\cleartabs` прямо перед `&`, когда желательно восстановить старое табулирование. Многобуквенные идентификаторы выглядят лучше, когда они помещены в тексте курсивом при помощи `\it`, как объяснено в главе 18. Поэтому рекомендуется следующее:

```
\+\bf while $p>0$ do\cr
  \+\quad\cleartabs&{\bf begin} $q:={\it link}(p)$;
    ${\it free\_node}(p)$; $p:=q$;\cr
  \+&{\bf end};\cr
```

22.3. Здесь мы применяем тот факт, что `&` вставляет новую позицию табуляции, когда справа от текущей позиции нет табуляторов. Надо изменить только одну макрокоманду, которая используется для обработки `\+`-строк, но (к сожалению) самую сложную:

```
\def\t@bb@x{\if@cr\egroup % теперь \box0 содержит колонку
  \else\hss\egroup \dimen@=0\p@
    \dimen@ii=\wd0 \advance\dimen@ii by1sp
    \loop\ifdim \dimen@<\dimen@ii
      \global\setbox\tabsyet=\hbox{\unhbox\tabsyet
        \global\setbox1=\lastbox}%
      \ifvoid1 \advance\dimen@ii by-\dimen@
        \advance\dimen@ii by-1sp \global\setbox1
          =\hbox to\dimen@ii{}\dimen@ii=-1pt\fi
      \advance\dimen@ by\wd1 \global\setbox\tabsdone
        =\hbox{\box1\unhbox\tabsdone}\repeat
    \setbox0=\hbox to\dimen@{\unhbox0}\fi
  \box0}
```

22.4. Горизонтальные списки Глава 14
Вертикальные списки Глава 15
Математические списки Глава 17 (т.е., первая колонка выровнена справа)

22.5.
```
Fowl&Poule de l'Ann\'ee&10 --- 12&Свыше 3&Тушить, Фрикассе\cr
```

22.6.
```
$$\halign to\hsize{\sl#\hfil\tabskip=.5em plus.5em&
  #\hfil\tabskip=0pt plus.5em&
  \hfil #\tabskip=1em plus2em&
```

```
\sl#\hfil\tabskip=.5em plus.5em&
#\hfil\tabskip=0pt plus.5em&
\hfil #\tabskip=0pt\cr ...}$$
```

22.7. Хитрость в том, чтобы определить новую макрокоманду для преамбулы:

```
$$\def\welshverb#1={{\bf#1} = }
\halign to\hsize{\welshverb#\hfil\tabskip=1em plus1em&
  \welshverb#\hfil&\welshverb#\hfil\tabskip=0pt\cr ...}$$
```

22.8.
```
\hfil#: &\vtop{\parindent=0pt\hsize=16em
    \hangindent.5em\strut#\strut}\cr
```

С такими узкими строками и такими длинными словами, возможно, внутри `\vtop` должен был быть увеличен `\tolerance` до, скажем 1000. К счастью, оказалось, что высокий допуск не нужен.

Замечание: Указанная преамбула решает проблему и демонстрирует, что способность TEX'а разбивать строки может быть использована внутри таблиц. Но эта конкретная таблица не очень хороший пример такого применения `\halign`, поскольку TEX мог напечатать ее непосредственно, используя подходящим образом `\everypar`, чтобы установить подвешенный отступ, и используя `\par` вместо `\cr`. Например, вы могли сказать

```
\hsize20em \parindent0pt \clubpenalty10000 \widowpenalty10000
\def\history#1&{\hangindent4.5em
  \hbox to4em{\hss#1: }\ignorespaces}
\everypar={\history} \def\\{\leavevmode{\it c\/}}
```

что избавляет TEX от всех работ по выравниванию `\halign` и дает по существу тот же самый результат.

22.9. Уравнения поделены на отдельные части для членов и знаков плюс/минус. Чтобы центрировать их, использован табличный клей:

```
$$\openup1\jot \tabskip=0pt plus1fil
\halign to\displaywidth{\tabskip=0pt
  $\hfil#$&$\hfil{}#{}$&
  $\hfil#$&$\hfil{}#{}$&
  $\hfil#$&$\hfil{}#{}$&
  $\hfil#$&${}#\hfil$\tabskip=0pt plus1fil&
  \llap{#}\tabskip=0pt\cr
10w&+&3x&+&3y&+&18z&=1,&(9)\cr
6w&-&17x&&&-&5z&=2.&(10)\cr}$$
```

22.10.
```
\hfil# &#\hfil&\quad#&\ \hfil#&\ \hfil#\cr
```

22.11.
```
\pmatrix{a_{11}&a_{12}&\ldots&a_{1n}\cr
    a_{21}&a_{22}&\ldots&a_{2n}\cr
    \multispan4\dotfill\cr
    a_{m1}&a_{m2}&\ldots&a_{mn}\cr}
```

22.12. В последней колонке, которая представляет собой вертикальную линию, `\cr` было бы опущено.

22.13. Один способ — это включить две строки непосредственно перед и после строки заголовка, сказав

```
\omit&height2pt&\multispan5&\cr.
```

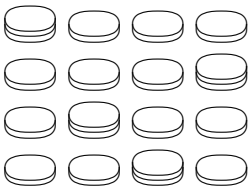
Другой способ — это поместить `\bigstrut` в некоторую колонку строки заголовка, чтобы получить некоторый подходящий невидимый бокс `\bigstrut` нулевой ширины. Любой из способов делает таблицу более симпатичной.

22.14. Хитрость заключается в том, чтобы получить "пустые" колонки слева и справа; в этом случае `\hrulefill` смогут проникнуть через табличный клей.

```
$$\vbox{\tabskip=0pt \offinterlineskip
\halign to 36em{\tabskip=0pt plus1em#&
  #\hfil&#&#\hfil&#&#\hfil&#\tabskip=0pt\cr
&&&&&\strut J. H. B\"ohning, 1838&\cr
&&&&\multispan3\hrulefill\cr
&&&\strut M. J. H. B\"ohning, 1882&\vrule\cr
&&\multispan3\hrulefill\cr
&&\vrule&&\vrule&\strut M. D. Blase, 1840&\cr
&&\vrule&&\multispan3\hrulefill\cr
&\strut L. M. Bohning, 1912&\vrule\cr
\multispan3\hrulefill\cr
&&\vrule&&&\strut E. F. Ehlert, 1845&\cr
&&\vrule&&\multispan3\hrulefill\cr
&&\vrule&\strut P. A. M. Ehlert, 1884&\vrule\cr
&&\multispan3\hrulefill\cr
&&&&\vrule&\strut C. L. Wischmeyer, 1850&\cr
&&&&\multispan3\hrulefill\cr}}$$
```

22.∞. (Решение проблемы Dudeney.) Пусть `\one` и `\two` — это макрокоманды, которые производят вертикальный список, обозначающий, соответственно, один или два пенса. Проблема может быть решена при помощи `\valign` следующим образом:

```
\valign{\vfil#&\vfil#&\vfil#&\vfil#\cr
  \two&\one&\one&\one\cr
  \one&\one&\two&\one\cr
  \one&\one&\one&\two\cr
  \one&\two&\one&\one\cr}
```



Так как `\valign` меняет ряды на колонки, получим .

23.1.
```
\footline={\hss\tenrm-- \folio\ --\hss}
```

23.2.
```
\headline={\ifnum\pageno=1 \hss\tenbf РЕЗЮМЕ\hss
  \else\tenrm Резюме A. B. Topa \dotfill\ Страница \folio\fi}
```

(Вы должны также сказать `\nopagenumbers` и `\voffset=2\baselineskip`.)

23.3.
```
\output={\plainoutput\blankpageoutput}
\def\blankpageoutput{\shipout\vbox{\makeheadline
    \vbox to\vsize{}\makefootline}\advancepageno}
```

23.4. Установите `\hsize=2.1in`, назначьте `\newbox\midcolumn` и используйте следующие команды:

```
\output={\if L\lr
    \global\setbox\leftcolumn=\columnbox \global\let\lr=M
  \else\if M\lr
    \global\setbox\midcolumn=\columnbox \global\let\lr=R
  \else \tripleformat \global\let\lr=L\fi\fi
  \ifnum\outputpenalty>-20000 \else\dosupereject\fi}
\def\tripleformat{\shipout\vbox{\makeheadline
    \fullline{\box\leftcolumn\hfil\box\midcolumn\hfil\columnbox}
    \makefootline}
  \advancepageno}
```

В конце поместите `\supereject` и дважды скажите

```
\if L\lr \else\null\vfill\eject\fi.
```

23.5. Т. У. Пицын забыл, что перед командой `\leftline` автоматически вставляется междустрочный клей. По правилам разбиения страниц в главе 15, это допускает законную точку разбиения между `\mark` и боксом `\leftline`. Можно было бы сказать `\nobreak` непосредственно после `\mark`, но обычно лучше поместить метки и вставки непосредственно *после* боксов.

23.6. Скажите, например, `\ifcase2\expandafter\relax\botmark\fi`, чтобы читать часть α_2 из `\botmark`. Другое решение помещает пять компонентов в пять параметров макрокоманды аналогично методу, который используется в этой главе позже командой `\inxcheck`. Подход `\ifcase` обычно более эффективен, поскольку позволяет TEX'у на высокой скорости пропустить невыбранные компоненты.

23.7.
```
\output={\dimen0=\dp255 \normaloutput
  \ifodd\pageno\else\if L\lr
    \expandafter\inxcheck\botmark\sub\end\fi\fi}
```

В этом случае макрокоманда `\normaloutput` должна быть двухколоночной программой вывода, которая описана в этой главе ранее, начинаясь с `\if L\lr` и оканчиваясь `\let\lr=L\fi`. (Здесь не нужна проверка `\supereject`.)

23.8. Не правильно. Если текст основного и/или вспомогательного элемента оказывается слишком длинным, строка продолжения в действительности может превратиться в две или более строк. (Между прочим, тогда будет подвешенный отступ, поскольку команда `\everypar`, которая задана вне программы вывода `\output`, действует и внутри ее.) Должен быть достаточно большой `\vsize`, чтобы вместить все строки продолжения плюс как минимум одну строку материала указателя, иначе может получиться бесконечный цикл.

24.1. Если `\cs` определена при помощи `\chardef` или `\mathchardef`, TEX, когда раскрывает `\meaning\cs`, использует шестнадцатиричную запись и присваивает категорию 12 каждой цифре расширения. У вас могла быть ситуация, в которой надо, чтобы последняя часть расширения рассматривалась как ⟨число⟩. (Это может быть скрытой причиной.)

24.2. Да; любому ключевому слову может предшествовать любое количество пробелов.

24.3. Первые две имеют одинаковое значение, а третья приводит `\baselineskip` к размеру, подавляя растяжимость или сжимаемость, которые могли присутствовать.

24.4. Естественная ширина равна 221 dd (которую TEX округляет до 15497423 sp и показывает как `236.47191pt`). Растяжимость равна 2500 sp, поскольку внутренние целые числа приводятся к размеру, который появляется как внутренняя единица. Сжимаемость равна нулю. Заметим, что последний `\space` поглощается как часть возможного пробела в части ⟨сжимаемость⟩ синтаксиса для ⟨клея⟩. (Если бы вместо `PLUS` был `MINUS`, последний `\space` *не* был бы частью этого клея!)

24.5. Если он был ненулевым, когда встретилось `\dump`. Приведем пример, который устанавливает `\batchmode` и помещает `\end` в конец входного файла:

```
\everyjob={\batchmode\input\jobname\end}
```

24.6. (a) `\def\\#1\\{}\futurelet\cs\\␣\\`. (b) `\def\\{\let\cs= }\\␣`. (Могут быть и другие решения.)

24.7. ⟨внутренняя величина⟩ ⟶ ⟨внутренняя величина⟩ | ⟨внутреннее целое⟩
| ⟨внутренний клей⟩ | ⟨внутренний mu клей⟩ | ⟨внутреннее нечисло⟩
⟨внутреннее нечисло⟩ ⟶ ⟨элементная переменная⟩ | ⟨шрифт⟩

26.1. Запись по основанию 10 используется для числовых констант и для вывода числовых данных. Первые 10 `\count`-регистров показываются при каждом `\shipout`, и в эти моменты их значения записываются в `dvi`-файл. Бокс, клей которого растянут или сжат на заданную растяжимость или сжимаемость, имеет плохость 100; эта плохость отделяет "свободные" боксы от "очень свободных" или "недополненных". TEX будет писать до 100 ошибок в одном абзаце, прежде чем остановиться (см. главу 27). Нормальное значение `\spacefactor` и `\mag` равно 1000. Значение `\prevdepth`, равное −1000 pt, подавляет междустрочный клей. Оценка плохости бокса имеет максимальное значение 10000. `INITEX` задает начальное значение `\tolerance`, равное 10000, поэтому делает выполнимыми все разбиения строк. Штраф 10000 или более запрещает разрыв, а штраф −10000 или менее делает принудительный разрыв. Стоимость разрыва страницы равна 100000, если плохость равна 10000 и если относящиеся к ней штрафы по величине меньше, чем 10000. (см. главу 15).

26.2. TEX позволяет, чтобы константы были выражены по основанию 8 (восьмеричная запись) и по основанию 16 (шестнадцатиричная запись). Он использует шестнадцатиричную запись, чтобы показать коды `\char` и `\mathchar`. Существует 16 семейств для математических шрифтов, 16 входных потоков для `\read`, 16 выходных потоков для `\write`. Значение `\catcode` должно быть меньше 16. Запись `^^?`, `^^@`, `^^A` указывает символы, код ASCII которых отличается на 64 от кодов `?`, `@`, `A`. Символы имеют код ASCII меньше, чем 128, следовательно, существуют по 128 элементов в каждой из таблиц `\catcode`, `\mathcode`, `\lccode`, `\uccode`, `\sfcode` и `\delcode`. Все значения `\lccode` и `\uccode` должны быть меньше, чем 128. Значение `\char` должно быть меньше, чем 256. Шрифт имеет не больше 256 символов. Существует 256 `\box`-регистров, 256 `\count`-регистров, 256 `\dimen`-регистров,

256 `\skip`-регистров, 256 `\muskip`-регистров, 256 `\toks`-регистров. "at-размер" шрифта должен быть меньше, чем 2048 pt, то есть 2^{11} pt. Математические ограничители закодированы умножением математического кода "маленького символа" на 2^{12}. Величина значения ⟨размер⟩ должна быть меньше, чем 16384 pt, то есть 2^{14} pt; аналогично, коэффициент ⟨factor⟩ в ⟨fil размер⟩ должны быть меньше, чем 2^{14}. Значение `\mathchar`, `\spacefactor` или `\sfcode` должны быть меньше, чем 2^{15}; значение `\mathcode` или `\mag` должны быть меньше или равны 2^{15}, а последнее значение обозначает "активный" математический символ. Единица `pt` равна 2^{16} sp . Значение `\delimiter` или `\delcode` должны быть меньше, чем 2^{27}. Команда `\end` иногда вносит на текущую страницу штраф, равный -2^{30}.⟨Размер⟩ по абсолютной величине должен быть меньше 2^{30} sp; ⟨Число⟩ по абсолютной величине должно быть меньше 2^{31}.

27.1. Он забыл посчитать пробел, поэтому TEX удалил "`i`", "`m`", "␣", "`\input`" и четыре буквы. (Но не все потеряно: можно напечатать "`1`" или "`2`", затем ⟨return⟩, и после подсказки "`*`" вводить новую строку.)

27.2. Сначала удалите нежелательные знаки, а затем вставьте нужные: введите "`6`", а затем "`I\macro`". (Между прочим, есть осторожный способ получить недоступную (`\inaccessible`) команду, печатая в ответ на такое сообщение об ошибке

```
I\garbage{}\let\accessible=
```

Автор создал TEX таким образом, что вы можете разрушить все, забавляясь с такими омерзительными трюками.)

27.3. Эту хитрость выполняет "`I␣%`", если `%` — символ комментария.

27.4. Часть слова "`minus`" в "`minuscule`" воспринялась, как часть команды `\hskip` в `\nextnumber`. Quick должен поместить в конце своей макрокоманды "`\relax`". (Ключевые слова `l`, `plus`, `minus`, `width`, `depth` или `height` могут встретиться в тексте, когда TEX читает спецификацию клея или линейки; конструктор макрокоманд общего пользования должен защищаться от этого. Если вы получаете ошибку '`Missing number`' и не можете догадаться, почему TEX ищет число, поместите инструкцию `\tracingcommands=1` где-нибудь перед ошибочной точкой. Тогда протокольный файл покажет, какие команды выполняет TEX.)

27.5. Если это упражнение не шутка, то заголовок этого приложения — обман.

Если вы не можете решить задачу,
вы всегда можете взглянуть на ответ.
Но, пожалуйста, постарайтесь решить
ее самостоятельно, тогда вы
научитесь большему и быстрее.

If you can't solve a problem,
you can always look up the answer.
But please, try first to solve it
by yourself; then you'll learn more
and you'll learn faster.

— DONALD E. KNUTH, *The TEXbook* (1983)

Как вы сами себе ответите?

How answer you for your selues?

— WILLIAM SHAKESPEARE, *Much Adoe About Nothing* (1598)

B

Основные командные последовательности

Начнем это приложение с обзора соглашений начального TEX'а.

Символы, резервированные для специальных целей: `\ { } $ & # % ^ _ ~`

```
\rm романский,{\sl наклонный},{\bf жирный},{\it курсивный\/} шрифт
```
романский, *наклонный*, **жирный**, *курсивный* шрифт

```
`` '' -- --- ?` !` \$ \# \& \% \ae \AE \oe \OE \aa \AA \ss \o \O
```
“ ” – — ¿ ¡ $ # & % æ Æ œ Œ å Å ß ø Ø

```
\`a \'e \^o \"u \=y \~n \.p \u\i \v s \H\j \t\i u \b k \c c \d h
```
à é ô ü ȳ ñ ṗ ĭ š ű̂ u̲ ç ẖ

```
\l \L \dag \ddag \S \P {\it\$ \&} \copyright \TeX \dots
```
ł Ł † ‡ § ¶ *£ &* © TEX …

Разбиение строк: `\break \nobreak \allowbreak \hbox{непереносимый}`
`воз\-мож\-ный пе\-ре\-нос` `\slash точка разрыва`

Разрываемые горизонтальные промежутки:
`\␣` нормальный пробел
`\enskip` такой пробел
`\quad` такой пробел
`\qquad` такой пробел
`\hskip` ⟨размер⟩

Неразрываемые горизонтальные промежутки:
`~` нормальный пробел
`\enspace` такой пробел
`\thinspace` такой пробел
`\negthinspace` такой пробел
`\kern` ⟨размер⟩

Вертикальные промежутки:
`\smallskip` `\medskip` `\bigskip`

Разбиение страниц: `\eject \supereject \nobreak \goodbreak \filbreak`
Вертикальные промежутки и хорошие разрывы:
`\smallbreak` `\medbreak` `\bigbreak`

```
\settabs 4 \columns
\+Вот&пример\hfill атрибутов &табулирования:&\hrulefill&\cr
```
Вот один пример табулирования: ———————

`\hrulefill` ——————— `\dotfill` .....

`\leftarrowfill` ←——— `\rightarrowfill` ———→

`\upbracefill` ⏟ `\downbracefill` ⏞

Сложные таблицы требуют `\halign`, `\valign`, `\omit`, `\span` и `\multispan`.

Примеры основных соглашений для макета текста на следующей странице.

```
% Этот файл создает  текст на соседней странице.
% Он сложный, поскольку показывает большой материал.
% TeX игнорирует комментарий (как этот) после  знака %.

% Сначала стандарт выходного стиля немного изменяется:
\hsize=29pc % Ширина строк этой книги равна 29 pc.
\vsize=42pc % Размер страницы равен 42 pc (без сноски).
\footline={\tenrm Нижний заголовок\quad\dotfill\quad стр. \folio}
\pageno=1009 % Номер страницы (не спрашивайте, почему).
% См. главу 23 для других изменений формата страницы с помощью
% \hoffset,\voffset,\nopagenumbers,\headline или \raggedbottom.

\vglue 1in % Это дает 1 дюйм пробела (1дюйм=2.54см).
\centerline{\bf Жирный заголовок в центре}
\smallskip % Маленький добавочный пробел после заголовка.
\rightline{\it avec un sous-titre \`a la fran\c caise}
% Теперь \beginsection  введет раздел 1 документа.
\beginsection 1. Plain \TeX нология % Следующая строка должна быть
пустой!

Первый абзац нового раздела не имеет отступа.
\TeX\ распознает конец абзаца, встречая пустую
строку входного файла. % или до \par: см. ниже.

Следующие абзацы {\it имеют\/} отступ.\footnote*{Величина
    отступа может быть изменена изменением параметра
{\tt\char`\\parindent}. Переверните страницу для обзора наиболее
важных параметров \TeX'а.} (Видите?) Компьютер разбивает абзац
на строки интересным способом --- см. ссылку~[1] --- и ав%
     томатически переносит слова,     если необходимо.

\midinsert % Это начинает вставку, например, рисунок.
\narrower\narrower % Это переносит поля (см. главу 14).
\noindent \llap{``}Если на текущей странице нет места для этого
 материала, он будет вставлен на следующей странице.''
\endinsert % Это оканчивает вставку и действие \narrower.

\proclaim Теорема T. Набор $математики$ обсуждается в
главах 16--19, а обзор математических символов  в приложении~F.

\beginsection 2. Библиография\par % \par --- как пустая строка.
\frenchspacing % (Глава 12 рекомендует это для библиографий.)
\item{[1]} D.~E. Knuth и M.~F. Plass, ``Breaking paragraphs
into lines,'' {\sl Softw. pract. exp. \bf11} (1981), 1119--1184.
\bye % Это способ закончить файл, не  командой \bang, а  \bye.
```

Жирный заголовок в центре

avec un sous-titre à la française

1. Plain TEXнология

Первый абзац нового раздела не имеет отступа. TEX распознает конец абзаца, когда доходит до пустой строки входного файла.

Следующие абзацы *имеют* отступ.* (Видите?) Компьютер разбивает абзац на строки интересным способом — см. ссылку [1] — и автоматически переносит слова, если необходимо.

> "Если на текущей странице нет места для этого материала,
> он будет вставлен на следующей."

Теорема T. *Набор* математики *обсуждается в главах 16–19, а обзор математических символов в приложении F.*

2. Библиография

[1] D. E. Knuth and M. F. Plass, "Breaking paragraphs into lines," *Softw. pract. exp.* **11** (1981), 1119–1184.

* Величина отступа может быть изменена параметром `\parindent`. Переверните страницу для обзора наиболее важных параметров TEX'а.

Нижний заголовок ..... стр. 1009

Предшествующий пример иллюстрирует то, что вы можете делать начальным TEX'ом, но не дает исчерпывающего списка возможностей. Так, были использованы `\centerline` и `\rightline`, но не `\leftline` и `\line`, `\midinsert`, но не `\topinsert` или `\pageinsert`, `\smallskip`, но не `\medskip` или `\bigskip`, `\llap`, но не `\rlap`, `\item`, но не `\itemitem`, `\vglue`, но не `\hglue`. Не показаны `\raggedright`-абзацы, не используется `\obeylines` или `\obeyspaces`, чтобы выключить автоматическое форматирование TEX'а. Все такие команды объясняются в этом приложении позднее, и дальнейшую информацию о них можно найти по предметному указателю. Основная цель приведенного примера — напомнить о наборе возможностей.

Большинство команд в примере являются макрокомандами формата начального TEX'а, но имеются и три примитива, то есть встроенные команды: `\par` (конец абзаца), `\noindent` (начало абзаца без отступа) и `\/` (курсивная поправка). В примере также присваиваются значения примитивным параметрам TEX'а `\hsize` и `\vsize`. TEX имеет множество параметров. Полный их список приводится в главе 24, но пользователей обычно касаются только несколько из них. Приведем пример, когда вы можете захотеть задать новые значения не `\hsize` и `\vsize`, а другим, более важным параметрам:

`\tolerance=500` (TEX разрешит строки, плохость которых не больше 500.)
`\looseness=1` (Следующий абзац будет строчкой длиннее обычного.)
`\parindent=4mm` (Отступ абзаца равен 4 миллиметра.)
`\hoffset=1.5in` (Результат сдвинут вправо на полтора дюйма.)
`\voffset=24pt` (Результат сдвинут вниз на 24 пункта.)
`\baselineskip=11pt plus.1pt` (Базовые линии отстоят на 11 pt или чуть больше.)
`\parskip=3pt plus1pt minus.5pt` (Перед каждым абзацем идет добавочный клей.)

Начальный TEX использует `\parindent` и для того, чтобы управлять величиной отступа, который задается командами `\item`, `\itemitem` и `\narrower`.

Остаток этого приложения посвящен деталям формата начального TEX'а, то есть набора макрокоманд, которые поставляются с нормальными реализациями TEX'а. Эти макрокоманды служат трем основным целям. (1) Они делают TEX удобным для использования, поскольку примитивные команды действуют на очень низком уровне. "Девственная" система TEX'а, которая не имеет макрокоманд, похожа на новорожденного младенца, которому надо необъятно много узнать о реальном мире, но он способен усвоить все накрепко. (2) Макрокоманды начального TEX'а обеспечивают базис для более совершенных и мощных форматов, приспособленных для индивидуальных вкусов и приложений. Можно многое сделать с помощью начального TEX'а, но очень скоро начинает хотеться еще большего. (3) Макрокоманды также служат для того, чтобы проиллюстрировать, как создавать дополнительные форматы.

Где-нибудь у себя вы должны найти файл `plain.tex`, который содержит то, что предварительно загружено в рабочую систему TEX'а, которую вы используете. В оставшейся части этого приложения нам надо обсудить содержание `plain.tex`. Однако, мы не будем заниматься дословным описанием этого файла, поскольку некоторые его части слишком однообразные и поскольку реальные макрокоманды "оптимизированы" в отношении размера памяти и времени работы. Для понимания более легкими являются неоптимизированные версии макрокоманд, поэтому мы будем иметь дело с ними. В файле `plain.tex` содержатся эквивалентные конструкции, которые работают лучше.

Итак, приведем план оставшейся части приложения B. Мы пройдем содержание файла `plain.tex`, сопровождая отредактированную копию комментариями по поводу заслуживающих внимания деталей. Когда мы подойдем к макрокоманде, использование которой еще не было объяснено — например, как-то получилось, что `\vglue` и `\beginsection` нигде в главах не упоминалось — мы будем рассматривать их с точки зрения пользователя. Но в большинстве случаев мы будем рассматривать команды с точки зрения конструктора макрокоманд.

1. Таблица кодов. Первая обязанность формата — установить значения `\catcode`. Это необходимо потому, например, что команда `\def` не может быть использована до тех пор, пока не существуют символы `{` и `}` категорий 1 и 2. Программа `INITEX` (которая читает `plain.tex`, так что TEX может быть инициализирован) начинает работу, не зная никаких символов группирования, следовательно, `plain.tex` начинает со следующего:

```
% This is the plain TeX format that's described in The TeXbook.
% N.B.: A version number is defined at the very end of this file;
%       please change that number whenever the file is modified!
% And don't modify the file under any circumstances.

\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\$=3 % dollar sign is math shift
\catcode`\&=4 % ampersand is alignment tab
\catcode`\#=6 % hash mark is macro parameter character
\catcode`\^=7 \catcode`\^^K=7 % circumflex and uparrow for superscripts
\catcode`\_=8 \catcode`\^^A=8 % underline and downarrow for subscripts
\catcode`\^^I=10 % ASCII tab is treated as a blank space
\chardef\active=13 \catcode`\~=\active % tilde is active
\catcode`\^^L=\active \outer\def^^L{\par} % ASCII form-feed is \outer\par

\message{Preloading the plain format: codes,}
```

Эти инструкции устанавливают для верхних и нижних индексов нестандартные символы `^^K` и `^^A` в дополнение к символам `^` и `_`, так что пользователи с расширенными наборами символов могут использовать ↑ и ↓, как рекомендовано в приложении C. Кроме того, символу `^^I` (⟨tab⟩ в ASCII) присваивается категория 10 (пробел), а `^^L` (⟨formfeed⟩ в ASCII) становится активным символом, который будет определять переход на файлы, поделенные на "страницы файла" символами ⟨formfeed⟩. Команда `\active` становится равной константе 13: это единственный номер категории, который заслуживает символического имени из-за частого использования для создания макрокоманд специального назначения.

Когда начинает работу INITEX, категория 12 (другие) присвоена всем 128 возможным символам, за тем исключением, что буквы имеют категорию 11 (буквы), и сделано несколько других присваиваний:

```
\catcode `\\ =0   \catcode`\  =10   \catcode `\% =14
\catcode`\^^@=9   \catcode`\^^M=5   \catcode`\^^?=15
```

Так, \ стал уже сигнальным символом, ␣ — пробелом, а % может задать комментарии уже на первой строке файла; ⟨null⟩ игнорируется, ⟨return⟩ является символом конца строки, а ⟨delete⟩ — ошибочным символом.

Когда plain.tex вводится при помощи INITEX, команда \message, показанная выше, выводит на терминал отчет о развитии событий. Позднее последует \message{registers,} и некоторые другие сообщения, но мы не будем упоминать их специально. Когда инициализация завершится, терминал покажет что-то вроде этого:

```
** plain
(plain.tex Preloading the plain format: codes, registers,
parameters, fonts, more fonts, macros, math definitions,
output routines, hyphenation (hyphen.tex))
* \dump
Beginning to dump on file plain.fmt
```

и далее разнообразную статистику о том, какие шрифты были загружены и так далее. Если вы хотите сделать новый формат super.tex, который добавляет дополнительные возможности к формату plain.tex, то, чтобы быстро ввести plain.fmt лучше не создавать новый файл, содержащий весь материал основного формата или \input plain, а просто ответить &plain super на подсказку ** программы INITEX.

После открывающего \message, plain.tex идет дальше и определяет команду \dospecials, которая перечисляет все символы, номер категории которых должен быть изменен на 12 (другие) при дословной печати:

```
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\^^K\do\_\do\^^A\do\%\do\~}
```

(Приложение E показывает, как использовать \dospecials.) Коды ASCII для ⟨null⟩, ⟨tab⟩, ⟨linefeed⟩, ⟨formfeed⟩, ⟨return⟩ и ⟨delete⟩ не включены в список.

В этот момент plain.tex завершает инициализацию номеров категорий, устанавливая \catcode`\@=11, после чего символ @ временно ведет себя как буква. Позже появится команда \catcode`\@=12, следовательно, at-символы во время работы TEX'а будут действовать как обычные знаки препинания. Идея в том, чтобы помочь начальному TEX'у иметь персональные команды, которые не могут переопределить обычные пользователи. У всех таких команд в имени будет как минимум один @.

Следующая забота — установить таблицу \mathcode:

```
\mathcode`\^^@="2201 \mathcode`\^^A="3223 \mathcode`\^^B="010B
\mathcode`\^^C="010C \mathcode`\^^D="225E \mathcode`\^^E="023A
\mathcode`\^^F="3232 \mathcode`\^^G="0119 \mathcode`\^^H="0115
\mathcode`\^^I="010D \mathcode`\^^J="010E \mathcode`\^^K="3222
```

```
\mathcode`\^^L="2206 \mathcode`\^^M="2208 \mathcode`\^^N="0231
\mathcode`\^^O="0140 \mathcode`\^^P="321A \mathcode`\^^Q="321B
\mathcode`\^^R="225C \mathcode`\^^S="225B \mathcode`\^^T="0238
\mathcode`\^^U="0239 \mathcode`\^^V="220A \mathcode`\^^W="3224
\mathcode`\^^X="3220 \mathcode`\^^Y="3221 \mathcode`\^^Z="8000
\mathcode`\^^[="2205 \mathcode`\^^\="3214 \mathcode`\^^]="3215
\mathcode`\^^^="3211 \mathcode`\^^_="225F \mathcode`\^^?="1273
\mathcode`\ ="8000   \mathcode`\!="5021   \mathcode`\'="8000
\mathcode`\(="4028   \mathcode`\)="5029   \mathcode`\*="2203
\mathcode`\+="202B   \mathcode`\,="613B   \mathcode`\-="2200
\mathcode`\.="013A   \mathcode`\/="013D   \mathcode`\:="303A
\mathcode`\;="603B   \mathcode`\<="313C   \mathcode`\=="303D
\mathcode`\>="313E   \mathcode`\?="503F   \mathcode`\[="405B
\mathcode`\\="026E   \mathcode`\]="505D   \mathcode`\_="8000
\mathcode`\{="4266   \mathcode`\|="026A   \mathcode`\}="5267
```

Математический код нужен только, если номер категории равен 11 или 12, поэтому многие из эти кодов будут редко нужны. Например, математический код для `^^M` задает символ `\oplus`, но трудно представить, что кому-то понадобился `^^M` (⟨return⟩ в ASCII) для знака $\oplus$ в середине формулы, поскольку начальный TEX добавляет `^^M` к каждой строке входного файла. Математические коды заданы, однако, полностью совместимыми с расширенным набором символов из приложения C и с Computer Modern шрифтами из приложения F. `INITEX` выполняет остальную работу, относящуюся к математическим кодам: устанавливает `\mathcode`$x = x + ''7000$ для каждой десятичной цифры $x =$ `'0` до `'9`; `\mathcode`$x = x + ''7100$ для каждой буквы и `\mathcode`$x = x$ для других x.

Менять таблицы `\uccode` и `\lccode` не надо. `INITEX` сделал `\uccode`\`X=\`X, `\uccode`\`x=\`X, `\lccode`\`X=\`x, `\lccode`\`x=\`x и аналогичные присваивания для всех других букв. Для всех небукв эти коды равны нулю. Эти таблицы используются TEX'ом для операций `\uppercase` и `\lowercase`. Алгоритм переноса также использует `\lccode` (см. приложение H). Изменения должны быть сделаны только в форматных пакетах, которые устанавливают TEX для языков, имеющих более 26 букв (см. главу 8).

Дальше идет таблица `\sfcode`, которую `INITEX` целиком установил в 1000, за исключением `\sfcode`\`X=999 для заглавных букв. Некоторые символы делаются "прозрачными":

```
\sfcode`\)=0 \sfcode`\'=0 \sfcode`\]=0 % won't change the space factor
```

а макрокоманда `\nonfrenchspacing` изменяет `\sfcode` специальных знаков препинания. (Глава 12 объясняет, что делает `\sfcode`.)

Последняя таблица называется `\delcode`, и снова надо изменить только несколько значений. `INITEX` делает коды всех ограничителей равными -1, что означает, что ни один символ в формулах не будет принят за ограничитель. Но есть исключение: предварительно определено значение `\delcode`\`\.=0, так что "." обозначает "нулевой ограничитель". (См. главу 17.) Начальный формат задает следующие девять значений, которые основаны на ограничителях, доступных в Computer Modern шрифтах:

```
\delcode`\(="028300    \delcode`\/="02F30E    \delcode`\)="029301
```

```
\delcode`\[="05B302    \delcode`\|="26A30C    \delcode`\]="05D303
\delcode`\<="26830A    \delcode`\\="26E30F    \delcode`\>="26930B
```

Важно отметить, что `\delcode`\{` и `\delcode`\}` остаются равными -1. Если бы эти коды были установлены в некоторое другое значение, пользователь мог бы напечатать, например, "`\big{`", чтобы получить большую левую фигурную скобку, что было бы большой ошибкой. Причина в том, что фигурные скобки используются для группирования, когда аргументы подаются в макрокоманды. Могут произойти всевозможные странности, если вы попытаетесь использовать их и как математические ограничители, и как ограничители группы.

В этом месте файл `plain.tex` содержит несколько определений

```
\chardef\@ne=1 \chardef\tw@=2 \chardef\thr@@=3 \chardef\sixt@@n=16
\chardef\@cclv=255 \mathchardef\@cclvi=256
\mathchardef\@m=1000 \mathchardef\@M=10000 \mathchardef\@MM=20000
```

которые позволяют "персональным" командам `\@ne`, `\tw@` и им подобным использоваться для констант 1, 2, ... Это позволяет TEX'у работать немного быстрее и приводит к тому, что макрокоманды требуют несколько меньшей памяти. Использование таких команд не будет, однако, показано без необходимости: мы будем притворяться, что вместо `\@ne` появляется "`1␣`", вместо `\@M` — "`10000␣`", и так далее, поскольку это делает программы более читабельными. (Заметим, что длинная форма `\@ne` есть "`1␣`", включая пробел, поскольку TEX ищет и убирает пробел, следующий за константой.)

2. Назначение регистров. Вторая важная часть файла `plain.tex` обеспечивает основание, на котором системы развивающихся макрокоманд могут мирно сосуществовать, не путаясь в использовании регистров. Авторы макрокоманд должны придерживаться следующих основополагающих правил. (1) Регистры с номерами от 0 до 9 всегда свободны для временных "пометок", но предполагается, что их значения разрушаются всякий раз, когда они попадают под действие другой макрокоманды. (Это относится к таким регистрам, как `\dimen0`, `\toks0`, `\skip1`, `\box3` и т.д., но регистры от `\count0` до `\count9` уже зарезервированы для идентификации номеров страниц.) (2) К регистрам `\count255`, `\dimen255` и `\skip255` доступ свободен в том же смысле. (3) Все присваивания регистров пометок, номера которых равны 1, 3, 5, 7 и 9, должны быть глобальными, а все присваивания другим регистрам пометок (0, 2, 4, 6, 8, 255) должны быть неглобальными. (Это предохраняет от "застройки стэка спасения", что обсуждалось в главе 27.) (4) Более того, можно использовать любой регистр в группе, если вы уверены, что механизм группирования TEX'а будет восстанавливать регистр, когда группа закончится, и что другие макрокоманды не сделают глобального присваивания этому регистру, когда он вам понадобится. (5) Но когда регистр используется несколькими макрокомандами или в течение длительного промежутка времени, присвоение ему должно быть сделано при помощи `\newcount`, `\newdimen`, `\newbox` и т.д. (6) Аналогичное примечание относится к входным/выходным потокам, которые используются командами `\read` и `\write`, к математическим семействам в команде семейства `\fam` и к вставкам, которые требуют, чтобы регистры `\box`, `\count`, `\dimen` и `\skip` все имели один и тот же номер.

В этом месте вводятся некоторые удобные аббревиатуры, так что регистры пометок будут легко доступны макрокомандам, приведенным ниже:

```
\countdef\count@=255  \toksdef\toks@=0       \skipdef\skip@=0
\dimendef\dimen@=0    \dimendef\dimen@i=1    \dimendef\dimen@ii=2
```

Далее идут макрокоманды, которые делают назначения величинам более постоянного значения. Для того, чтобы содержать числа, которые назначены только что, эти макрокоманды используют регистры от `\count10` до `\count19`. Например, если `\newdimen` только что зарезервировал `\dimen15`, значение `\count11` будет 15. Однако, остальной мир может не "знать", что `\count11` должен что-то делать с регистром `\dimen`. Существует специальный счетчик, называемый `\allocationnumber`, который будет равен самому последнему назначенному номеру после каждой операции `\newcount`, `\newdimen`, ..., `\newinsert`. Предполагается, что макропакеты ссылаются на `\allocationnumber`, если им надо найти, какой номер был назначен. Оказывается, что `\allocationnumber` — это `\count20`, но другие пакеты не знают этого. Другими словами, история того, как в действительности происходят назначения, не должна иметь отношения к делу, когда макрокоманды назначения используются на высоком уровне. Вам не надо знать, какие назначения в действительности делает `plain.tex` в каждом конкретном случае.

```
\count10=21 % this counter allocates \count registers 22, 23, 24, ...
\count11=9  % this counter allocates \dimen registers 10, 11, 12, ...
\count12=9  % this counter allocates \skip registers 10, 11, 12, ...
\count13=9  % this counter allocates \muskip registers 10, 11, 12, ...
\count14=9  % this counter allocates \box registers 10, 11, 12, ...
\count15=9  % this counter allocates \toks registers 10, 11, 12, ...
\count16=-1 % this counter allocates input streams 0, 1, 2, ...
\count17=-1 % this counter allocates output streams 0, 1, 2, ...
\count18=3  % this counter allocates math families 4, 5, 6, ...
\count19=255 % this counter allocates insertions 254, 253, 252, ...
\countdef\insc@unt=19 % nickname for the insertioncounter
\countdef\allocationnumber=20 % the most recent allocation
\countdef\m@ne=21 \m@ne=-1 % a handy constant
\def\wlog{\immediate\write-1} % this will write on log file (only)

\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\outer\def\newskip{\alloc@2\skip\skipdef\insc@unt}
\outer\def\newmuskip{\alloc@3\muskip\muskipdef\@cclvi}
\outer\def\newbox{\alloc@4\box\chardef\insc@unt}
\let\newtoks=\relax % this allows plain.tex to be read in twice
\outer\def\newhelp#1#2{\newtoks#1#1=\expandafter{\csname#2\endcsname}}
\outer\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
\outer\def\newread{\alloc@6\read\chardef\sixt@@n}
\outer\def\newwrite{\alloc@7\write\chardef\sixt@@n}
\outer\def\newfam{\alloc@8\fam\chardef\sixt@@n}

\def\alloc@#1#2#3#4#5{\global\advance\count1#1 by 1
  \ch@ck#1#4#2% make sure there's still room
  \allocationnumber=\count1#1
```

```
  \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}
\outer\def\newinsert#1{\global\advance\insc@unt by-1
  \ch@ck0\insc@unt\count \ch@ck1\insc@unt\dimen
  \ch@ck2\insc@unt\skip  \ch@ck4\insc@unt\box
  \allocationnumber=\insc@unt
  \global\chardef#1=\allocationnumber
  \wlog{\string#1=\string\insert\the\allocationnumber}}
\def\ch@ck#1#2#3{\ifnum\count1#1<#2%
  \else\errmessage{No room for a new #3}\fi}
```

Макрокоманда `\alloc@` выполняет основную работу по назначению. Она помещает сообщение типа `\maxdimen=\dimen10` в протокольный файл после того, как команда `\newdimen` назначила место для регистра `\dimen`, который теперь будет называться `\maxdimen`. Такая информация может быть полезной, когда отлаживаются трудные макрокоманды .

Макрокоманда `\newhelp` оказывает помощь при создании самодельных вспомогательных сообщений. Например, можно сказать `\newhelp\helpout{Это --- вспомогательное сообщение.}` и затем дать команду `\errhelp=\helpout` непосредственно перед `\errmessage`. Этот метод создания вспомогательных текстов эффективно использует память TEX'а, поскольку помещает текст в имя команды, где он не занимает места, необходимого для элементов.

Далее файл `plain` назначает регистры для важных констант

```
\newdimen\maxdimen \maxdimen=16383.99999pt
\newskip\hideskip \hideskip=-1000pt plus1fill
\newskip\centering \centering=0pt plus 1000pt minus 1000pt
\newdimen\p@ \p@=1pt % this saves macro space and time
\newdimen\z@ \z@=0pt \newskip\z@skip \z@skip=0pt plus0pt minus0pt
\newbox\voidb@x % permanently void box register
```

Команда `\maxdimen` обозначает наибольший допустимый ⟨размер⟩. Макрокоманды выравнивания, которые появляются ниже, будут использовать специальные клеи `\hideskip` и `\centering`. *N.B.: Эти три константы не должны изменяться ни при каких обстоятельствах.* Вы должны либо полностью игнорировать их, либо использовать и радоваться. Фактически, следующим четырем регистрам констант (`\p@`, `\z@`, `\z@skip` и `\voidb@x`) даны личные имена, так что они неприкосновенны. Команда `\p@` используется десятки раз как аббревиатура для "`pt `", а `\z@` очень часто используется для обозначения "`0pt `" или "`0 `". Использование таких аббревиатур экономит почти 10% памяти, необходимой для хранения элементов в макрокомандах начального TEX'а. Но ниже мы будем показывать не сокращенные формы, так что макрокоманды будут более читабельными.

Далее следуют различного сорта назначения:

```
\outer\def\newif#1{\count@=\escapechar \escapechar=-1
  \expandafter\expandafter\expandafter
   \edef\@if#1{true}{\let\noexpand#1=\noexpand\iftrue}%
  \expandafter\expandafter\expandafter
   \edef\@if#1{false}{\let\noexpand#1=\noexpand\iffalse}%
```

```
  \@if#1{false}\escapechar=\count@} % the condition starts out false
\def\@if#1#2{\csname\expandafter\if@\string#1#2\endcsname}
{\uccode`1=`i \uccode`2=`f \uppercase{\gdef\if@12{}}} % `if' is required
```

Например, команда `\newif\ifalpha` создает три команды `\alphatrue`, `\alphafalse` и `\ifalpha` (см. главу 20).

3. Параметры. `INITEX` устанавливает почти все числовые регистры и параметры в нуль, все регистры элементов и параметры делает пустыми, а также делает пустыми все регистры боксов. Но есть несколько исключений: `\mag` изначально устанавливается в 1000, `\tolerance` в 10000, `\maxdeadcycles` в 25, `\hangafter` в 1, `\escapechar` в `` `\\ `` и `\endlinechar` в `` `\^^M ``. Начальный TEX присваивает параметрам следующие значения:

```
\pretolerance=100 \tolerance=200  \hbadness=1000 \vbadness=1000
\linepenalty=10 \hyphenpenalty=50 \exhyphenpenalty=50
\binoppenalty=700 \relpenalty=500
\clubpenalty=150 \widowpenalty=150 \displaywidowpenalty=50
\brokenpenalty=100 \predisplaypenalty=10000
\doublehyphendemerits=10000 \finalhyphendemerits=5000 \adjdemerits=10000
\tracinglostchars=1 \uchyph=1 \delimiterfactor=901
\defaulthyphenchar=`\- \defaultskewchar=-1 \newlinechar=-1
\showboxbreadth=5 \showboxdepth=3

\hfuzz=0.1pt \vfuzz=0.1pt \overfullrule=5pt
\hsize=6.5in \vsize=8.9in \parindent=20pt
\maxdepth=4pt \splitmaxdepth=\maxdimen \boxmaxdepth=\maxdimen
\delimitershortfall=5pt \nulldelimiterspace=1.2pt \scriptspace=0.5pt

\parskip=0pt plus 1pt
\abovedisplayskip=12pt plus 3pt minus 9pt
\abovedisplayshortskip=0pt plus 3pt
\belowdisplayskip=12pt plus 3pt minus 9pt
\belowdisplayshortskip=7pt plus 3pt minus 4pt
\topskip=10pt \splittopskip=10pt
\parfillskip=0pt plus 1fil

\thinmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
```

(Некоторые параметры устанавливаются самим TEX'ом во время работы, так что их не следует инициализировать: `\time`, `\day`, `\month` и `\year` устанавливаются в начале работы, параметру `\outputpenalty` дается значение, когда вызывается программа вывода, `\predisplaysize`, `\displaywidth` и `\displayindent` получают значение перед выделением, а значения `\looseness=0`, `\hangindent=0pt`, `\hangafter=1` и `\parshape=0` присваиваются в конце абзаца и когда TEX входит во внутреннюю вертикальную моду.)

Параметры `\baselineskip`, `\lineskip` и `\lineskiplimit` здесь не инициализированы, но ниже будет определена макрокоманда `\normalbaselines`, которая устанавливает `\baselineskip=\normalbaselineskip`, `\lineskip=\normallineskip` и

`\lineskiplimit=\normallineskiplimit`. При таком непрямом подходе можно управлять несколькими различными типами размеров, как показано в приложении E. Начальный TEX имеет дело исключительно с шрифтами в 10 пунктов, но поддерживает расширение на другие стили.

Далее идут несколько "псевдопараметров". Эти величины ведут себя так же, как внутренние параметры TEX'а, и пользователи могут менять их тем же способом, но они скорее часть формата начального TEX'а, чем примитивные операторы языка.

```
\newskip\smallskipamount % the amount of a \smallskip
  \smallskipamount=3pt plus1pt minus1pt
\newskip\medskipamount % the amount of a \medskip
  \medskipamount=6pt plus2pt minus2pt
\newskip\bigskipamount % the amount of a \bigskip
  \bigskipamount=12pt plus4pt minus4pt
\newskip\normalbaselineskip % normal value of \baselineskip
  \normalbaselineskip=12pt
\newskip\normallineskip % normal value of \lineskip
  \normallineskip=1pt
\newdimen\normallineskiplimit % normal value of \lineskiplimit
  \normallineskiplimit=0pt
\newdimen\jot % unit of measure for openning up displays
  \jot=3pt
\newcount\interdisplaylinepenalty % interline penalty in \displaylines
  \interdisplaylinepenalty=100
\newcount\interfootnotelinepenalty %  interline penalty in footnotes
  \interfootnotelinepenalty=100
```

4. Информация о шрифтах. Теперь `plain.tex` вносит данные, которые нужны TEX'у, чтобы знать, как печатать символы шрифтов. Сначала определяются макрокоманды `\magstep`, чтобы поддерживать масштаб шрифта:

```
\def\magstephalf{1095 }
\def\magstep#1{\ifcase#1 1000\or
  1200\or 1440\or 1728\or 2074\or 2488\fi\relax}
```

(Между прочим, команда `\magstep` не использует `\multiply` для вычисления значений, поскольку предполагается, что она раскрывается в ⟨число⟩ по пути в "желудок" TEX'а. `\multiply` не работала бы, поскольку это команда присваивания, выполняемая только в желудке.)

Главное, что отличает один формат от другого — это тот факт, что каждый формат дает TEX'у необходимые знания о некоторых семействах шрифтов. В нашем случае за основу берутся Computer Modern шрифты, описанные в приложении F, хотя есть условия для присоединения и других стилей.

```
\font\tenrm=cmr10       \font\preloaded=cmr9     \font\preloaded=cmr8
\font\sevenrm=cmr7      \font\preloaded=cmr6     \font\fiverm=cmr5

\font\teni=cmmi10       \font\preloaded=cmmi9    \font\preloaded=cmmi8
\font\seveni=cmmi7      \font\preloaded=cmmi6    \font\fivei=cmmi5
```

```
\font\tensy=cmsy10       \font\preloaded=cmsy9    \font\preloaded=cmsy8
\font\sevensy=cmsy7      \font\preloaded=cmsy6    \font\fivesy=cmsy5
\font\tenex=cmex10
\font\tenbf=cmbx10       \font\preloaded=cmbx9    \font\preloaded=cmbx8
\font\sevenbf=cmbx7      \font\preloaded=cmbx6    \font\fivebf=cmbx5
\font\tensl=cmsl10       \font\preloaded=cmsl9    \font\preloaded=cmsl8
\font\tentt=cmtt10       \font\preloaded=cmtt9    \font\preloaded=cmtt8
\font\tenit=cmti10       \font\preloaded=cmti9    \font\preloaded=cmti8
\font\preloaded=cmss10   \font\preloaded=cmssq8
\font\preloaded=cmssi10  \font\preloaded=cmssqi8
\font\preloaded=cmr7 scaled \magstep4 % for titles
\font\preloaded=cmtt10 scaled \magstep2
\font\preloaded=cmssbx10 scaled \magstep2
% Additional \preloaded fonts can be specified here.
% (And those that were \preloaded above can be eliminated.)
\let\preloaded=\undefined % preloaded fonts must be declared anew later.
```

Заметим, что большинство шрифтов названы `\preloaded`, но команда `\preloaded` в самом конце сделана неопределенной, так что эти шрифты не могут быть использованы непосредственно. Для такого странного подхода есть две причины: во-первых, желательно, чтобы общее число шрифтов начального TEX'а было относительно маленьким, поскольку начальный TEX представляет собой стандартный формат, и для пользователя не должно быть дорого присоединять все шрифты начального TEX'а к тем, которые ему действительно нужны. Во-вторых, на многих компьютерных системах желательно предварительно загружать информацию для большинства шрифтов, которыми часто будут пользоваться, поскольку это экономит машинное время. Информация о `\preloaded`-шрифте поступает в память TEX'а, где она моментально оживет, как только пользователь снова определит соответствующий `\font`. Например, пример формат книги в приложении E говорит `\font\ninerm=cmr9`. После того, как выполнено это присваивание, команда `\ninerm` будет идентифицировать шрифт, информация о котором не должна снова загружаться.

Точное число и природа предварительно загружаемых шрифтов не имеет значения. Единственное, что необходимо для стандартизации между машинами — чтобы действительно были загружены шестнадцать основных шрифтов (`cmr10`, `cmr7`, ..., `cmti10`). Файлы `plain.tex`, используемые на различных машинах, могут быть различными по отношению к предварительно загружаемым шрифтам, поскольку выбор того, сколько надо загружать шрифтов и отбор наиболее важных из них зависит от местных условий. Например, в университете автора полезно загрузить шрифт, который содержит печать Стэнфорда, но этот шрифт не очень популярен в Беркли.

Большинство из этих шрифтов имеют задаваемые по умолчанию значения `\hyphenchar` и `\skewchar`, а именно, `'-` и `-1`, но математический курсивный и математический символьный шрифты имеют специальные значения `\skewchar`, которые определяются так:

```
\skewchar\teni='177 \skewchar\seveni='177 \skewchar\fivei='177
\skewchar\tensy='60 \skewchar\sevensy='60 \skewchar\fivesy='60
```

После загрузки шрифты группируются в семейства, чтобы их можно было задавать при наборе математики, и определяются такие сокращенные имена, как `\rm` и `\it`:

```
\textfont0=\tenrm \scriptfont0=\sevenrm \scriptscriptfont0=\fiverm
\def\rm{\fam0 \tenrm}
\textfont1=\teni  \scriptfont1=\seveni  \scriptscriptfont1=\fivei
\def\mit{\fam1 } \def\oldstyle{\fam1 \teni}
\textfont2=\tensy \scriptfont2=\sevensy \scriptscriptfont2=\fivesy
\def\cal{\fam2 }
\textfont3=\tenex \scriptfont3=\tenex   \scriptscriptfont3=\tenex
\newfam\itfam \def\it{\fam\itfam\tenit} \textfont\itfam=\tenit
\newfam\slfam \def\sl{\fam\slfam\tensl} \textfont\slfam=\tensl
\newfam\bffam \def\bf{\fam\bffam\tenbf} \textfont\bffam=\tenbf
  \scriptfont\bffam=\sevenbf   \scriptscriptfont\bffam=\fivebf
\newfam\ttfam \def\tt{\fam\ttfam\tentt} \textfont\ttfam=\tentt
```

5. Макрокоманды для текста. Пятая часть файла `plain.tex` вводит макрокоманды, с помощью которых делается основное форматирование текста, не относящееся к математике. Сначала идут несколько команд, которые были обещаны выше:

```
\def\frenchspacing{\sfcode`\.=1000 \sfcode`\?=1000 \sfcode`\!=1000
  \sfcode`\:=1000 \sfcode`\;=1000 \sfcode`\,=1000 }
\def\nonfrenchspacing{\sfcode`\.=3000 \sfcode`\?=3000 \sfcode`\!=3000
  \sfcode`\:=2000 \sfcode`\;=1500 \sfcode`\,=1250 }
\def\normalbaselines{\lineskip=\normallineskip
  \baselineskip=\normalbaselineskip \lineskiplimit=\normallineskiplimit}
```

Следующие несколько команд просты, но очень важны. Сначала \⟨tab⟩ и \⟨return⟩ определяются так, что они раскрываются в \⟨space⟩. Это помогает избежать путаницы, поскольку все три случая на большинстве терминалов показываются одинаково. Затем определяются макрокоманды `\lq`, `\rq`, `\lbrack` и `\rbrack` для тех, у кого есть трудности со знаками кавычек и/или скобок. Команды `\endgraf` и `\endline` сделаны эквивалентными примитивным операциям TEX'а `\par` и `\cr`, поскольку часто полезно переопределить значения самих `\par` и `\cr`. Затем следуют определения `\space` (чистый промежуток), `\empty` (список без элементов) и `\null` (пустой h-бокс). Наконец, сделаны `\bgroup` и `\egroup`, чтобы обеспечить символы "неявного" группирования, которые оказываются особенно полезными в макроопределениях. (Информация о неявных символах в главах 24–26 и в приложение D.)

```
\def\^^I{\ } \def\^^M{\ }
\def\lq{`} \def\rq{'} \def\lbrack{[} \def\rbrack{]}
\let\endgraf=\par \let\endline=\cr
\def\space{ } \def\empty{} \def\null{\hbox{}}
\let\bgroup={ \let\egroup=}
```

Теперь в определениях `\obeyspaces` и `\obeylines` возникает несколько хитростей, поскольку TEX только "наполовину послушен", пока эти определения наполовину закончены:

```
\def\obeyspaces{\catcode`\ =\active}
{\obeyspaces\global\let =\space}
{\catcode`\^^M=\active % these lines must end with `%'
  \gdef\obeylines{\catcode`\^^M=\active \let^^M=\par}%
  \global\let^^M=\par} % this is in case ^^M appears in a \write
```

Макрокоманда `\obeylines` говорит `\let^^M=\par` вместо `\def^^M{\par}`, поскольку `\let` позволяет такие конструкции, как `\let\par=\cr \obeylines \halign{...}`, в которых `\cr` заданы не внутри выравнивания.

Макрокоманда `\loop...\repeat` обеспечивает повторяющиеся операции, как показано в главе 20. В этой и некоторых других макрокомандах команде `\next` дано временное значение, которое далее не нужно. Таким образом, `\next` действует как "команда для пометок".

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next=\iterate \else\let\next=\relax\fi \next}
\let\repeat=\fi % this makes \loop...\if...\repeat skippable
```

Следующая забота о пробелах. Макрокоманды `\enskip`, `\quad` и `\qquad` делают пробелы, которые являются законными точками разрыва внутри абзаца, `\enspace`, `\thinspace` и `\negthinspace` производят пробелы, на которых не может быть разрыва (хотя пробел будет исчезать, если он встречается непосредственно перед некоторыми видами разрывов). Все шесть этих пробелов зависят от текущего шрифта. Вы можете получить горизонтальный пробел, который никогда не исчезает, если скажите `\hglue`⟨клей⟩. Этот пробел может растягиваться или сжиматься. Аналогично, существует его вертикальный аналог `\vglue`⟨клей⟩. Макрокоманда `\nointerlineskip` подавляет междустрочный клей, который обычно был бы вставлен в вертикальной моде перед следующим боксом. Это "одноразовая" макрокоманда, но есть и более сильнодействующая команда `\offinterlineskip` — она устраивает так, что будущий междустрочный клей присутствует, но равен нулю. Есть также макрокоманды для потенциально разрываемых вертикальных пробелов: `\smallskip`, `\medskip` и `\bigskip`.

```
\def\enskip{\hskip.5em\relax}
\def\quad{\hskip1em\relax} \def\qquad{\hskip2em\relax}
\def\enspace{\kern.5em }
\def\thinspace{\kern .16667em } \def\negthinspace{\kern-.16667em }

\def\hglue{\afterassignment\hgl@\skip@=}
\def\hgl@{\leavevmode \count@=\spacefactor \vrule width0pt
  \nobreak\hskip\skip@ \spacefactor=\count@}
\def\vglue{\afterassignment\vgl@\skip@=}
\def\vgl@{\par \dimen@=\prevdepth \hrule height0pt
  \nobreak\vskip\skip@ \prevdepth=\dimen@}

\def\nointerlineskip{\prevdepth=-1000pt }
\def\offinterlineskip{\baselineskip=-1000pt
  \lineskip=0pt \lineskiplimit=\maxdimen}
```

```
\def\smallskip{\vskip\smallskipamount}
\def\medskip{\vskip\medskipamount}
\def\bigskip{\vskip\bigskipamount}
```

Продолжим разговор о точках разрыва. Следующие макрокоманды вводят штрафные маркеры, которые делают разрыв более или менее желательным. Макрокоманды `\break`, `\nobreak` и `\allowbreak` работают в любой моде, макрокоманды `~` (связка) и `\slash` (подобный дефису '/') работают в горизонтальной моде, а остальные работают только в вертикальной моде, то есть между абзацами, поэтому они начинаются с `\par`.

```
\def\break{\penalty-10000 } \def\nobreak{\penalty10000 }
\def\allowbreak{\penalty0 }
\def~{\penalty10000\ }
\def\slash{/\penalty\exhyphenpenalty}

\def\filbreak{\par\vfil\penalty-200\vfilneg}
\def\goodbreak{\par\penalty-500 }
\def\eject{\par\penalty-10000 }
\def\supereject{\par\penalty-20000 }

\def\removelastskip{\ifdim\lastskip=0pt \else\vskip-\lastskip\fi}
\def\smallbreak{\par \ifdim\lastskip<\smallskipamount
  \removelastskip \penalty-50 \smallskip \fi}
\def\medbreak{\par \ifdim\lastskip<\medskipamount
  \removelastskip \penalty-100 \medskip \fi}
\def\bigbreak{\par \ifdim\lastskip<\bigskipamount
  \removelastskip \penalty-200 \bigskip \fi}
```

Дальше следуют боксы: `\line`, `\leftline`, `\rightline` и `\centerline` делают боксы, ширина которых равна полной строке, в то время как `\llap` и `\rlap` создают боксы, эффективная ширина которых равна нулю. Макрокоманда `\underbar` помещает свой аргумент в h-бокс с прямой линией, расположенной на фиксированном расстоянии под этим аргументом.

```
\def\line{\hbox to\hsize}
\def\leftline#1{\line{#1\hss}} \def\rightline#1{\line{\hss#1}}
\def\centerline#1{\line{\hss#1\hss}}

\def\llap#1{\hbox to 0pt{\hss#1}} \def\rlap#1{\hbox to 0pt{#1\hss}}

\def\m@th{\mathsurround=0pt }
\def\underbar#1{$\setbox0=\hbox{#1} \dp0=0pt
  \m@th \underline{\box0}$}
```

(Заметим, что `\underbar` в своей работе использует математическую моду, хотя эта операция, по существу, имеет нематематическую природу. Аналогичным образом используют математическую моду и несколько других макрокоманд, приведенных ниже. Таким образом, математические возможности TEX'а оказываются полезными, даже когда нет никакой математики. Для того, чтобы "выключить" `\mathsurround` при выполнении таких конструкций, используется специальная команда `\m@th`.)

Здесь `\strut` реализуется, как линейка нулевой ширины, поскольку это занимает минимум места и времени в тех приложениях, в которых присутствуют многочисленные подпорки.

```
\newbox\strutbox
\setbox\strutbox=\hbox{\vrule height8.5pt depth3.5pt width0pt}
\def\strut{\relax\ifmmode\copy\strutbox\else\unhcopy\strutbox\fi}
```

В том случае, когда в этой и в других приведенных ниже макрокомандах в элементе выравнивания первым появляется `\strut`, необходима команда `\relax`, поскольку в это время TEX находится в какой-нибудь непредсказуемой моде (см. главу 22).

Макрокоманда `\ialign` предусмотрена для выравниваний, когда необходима уверенность, что значение `\tabskip` первоначально равно нулю. Макрокоманда `\hidewidth` может быть использована как `\hfill` в тех элементах выравниваний, которым разрешено "высовываться" из своих колонок. Имеется также `\multispan`, которая позволяет элементам выравнивания находиться в одной или более колонок.

```
\def\ialign{\everycr={}\tabskip=0pt \halign} % инициализировано \halign
\def\hidewidth{\hskip\hideskip}

\newcount\mscount
\def\multispan#1{\omit \mscount=#1 \loop\ifnum\mscount>1 \sp@n\repeat}
\def\sp@n{\span\omit \advance\mscount by -1 }
```

Далее мы получаем "табулирующие" макрокоманды, которые сложнее всего, что есть в начальном TEX'е. Они сохраняют позиции табуляции, поддерживая боксы, которые заполнены пустыми боксами, имеющими указанную ширину. (Для того, чтобы понять эти макрокоманды, лучше всего понаблюдать их в действии на простых примерах, используя `\tracingall`.)

```
\newif\ifus@ \newif\if@cr
\newbox\tabs \newbox\tabsyet \newbox\tabsdone

\def\cleartabs{\global\setbox\tabsyet=\null \setbox\tabs=\null}
\def\settabs{\setbox\tabs=\null \futurelet\next\sett@b}
\let\+=\relax % in case this file is being read in twice
\def\sett@b{\ifx\next\+ \let\next=\relax % turn off \outerness
    \def\next{\afterassignment\s@tt@b\let\next}%
  \else\let\next=\s@tcols\fi\next}
\def\s@tt@b{\let\next=\relax \us@false\m@ketabbox}
\outer\def\+{\tabalign} \def\tabalign{\us@true \m@ketabbox}
\def\s@tcols#1\columns{\count@=#1 \dimen@=\hsize
  \loop \ifnum\count@>0 \@nother \repeat}
\def\@nother{\dimen@ii=\dimen@ \divide\dimen@ii by\count@
  \setbox\tabs=\hbox{\hbox to\dimen@ii{}\unhbox\tabs}%
  \advance\dimen@ by-\dimen@ii \advance\count@ by -1 }

\def\m@ketabbox{\begingroup
  \global\setbox\tabsyet=\copy\tabs \global\setbox\tabsdone=\null
  \def\cr{\@crtrue\crcr\egroup\egroup
```

```
    \ifus@ \unvbox0 \lastbox\fi \endgroup
    \setbox\tabs=\hbox{\unhbox\tabsyet\unhbox\tabsdone}}%
  \setbox0=\vbox\bgroup\@crfalse \ialign\bgroup&\t@bbox##\t@bb@x\crcr}
\def\t@bbox{\setbox0=\hbox\bgroup}
\def\t@bb@x{\if@cr\egroup % now \box0 holds the column
  \else\hss\egroup \global\setbox\tabsyet=\hbox{\unhbox\tabsyet
      \global\setbox1=\lastbox}% now \box1 hold its size
    \ifvoid1 \global\setbox1=\hbox to\wd0{}%
    \else\setbox0=\hbox to\wd1{\unhbox0}\fi
    \global\setbox\tabsdone=\hbox{\box1\unhbox\tabsdone}\fi
  \box0}
```

Здесь макрокоманда `\+` объявлена `\outer`, так что TEX'у будет легче обнаружить убегающие аргументы и определения (см. главу 20). В том случае, когда необходимо использовать `\+` в некотором "внутреннем" месте, также предусмотрена не-`\outer` версия, называемая `\tabalign`. Можно использовать `\tabalign` точно также, как `\+`, но только не после `\settabs`.

• Формы абзацев ограниченного, но важного вида обеспечена командами `\item`, `\itemitem` и `\narrower`. Также есть две макрокоманды, которые не упоминались прежде. (1) Команда `\hang` служит причиной подвешенного отступа нормальной величины `\parindent` после первой строки, таким образом, во всем абзаце будет сделан отступ одной и той же величины (если только он не начинался с `\noindent`). (2) Команда `\textindent{stuff}` такая же, как `\indent`, только помещает "`stuff`" в отступ, отступив от его правого края на один en влево. Она также убирает те пробелы, которые могли следовать за правой фигурной скобкой в `{stuff}`. Например, настоящий абзац был напечатан командой "`\textindent{$\bullet$} Формы абзацев ...`". Первая "Ф" появляется в обычной для первой буквы абзаца позиции.

```
\def\hang{\hangindent\parindent}
\def\item{\par\hang\textindent}
\def\itemitem{\par\indent \hangindent2\parindent \textindent}
\def\textindent#1{\indent\llap{#1\enspace}\ignorespaces}
\def\narrower{\advance\leftskip by\parindent
  \advance\rightskip by\parindent}
```

Макрокоманда `\beginsection` предназначена, чтобы отметить начало нового главного подразделения в документе. Чтобы использовать ее, вы вслед за пустой строкой (или за `\par`) должны сказать `\beginsection`⟨имя секции⟩. Макрокоманда сначала выделяет клей и штрафы, обеспечивающие начало новой страницы, если настоящая страница почти заполнена, затем она делает `\bigskip` и печатает имя секции жирным шрифтом на отдельной строке, прижав его влево. Имя секции также показывается на терминале. После `\smallskip` с запрещенным разрывом страницы дается команда `\noindent`, что подавляет отступ в следующем абзаце, то есть первом абзаце новой секции. (Однако, если за командой `\beginsection` сразу следует вертикальный материал, следующий "абзац" будет пустым.)

```
\outer\def\beginsection#1\par{\vskip0pt plus.3\vsize\penalty-250
```

```
  \vskip0pt plus-.3\vsize\bigskip\vskip\parskip
  \message{#1}\leftline{\bf#1}\nobreak\smallskip\noindent}
```

В математических статьях специальные утверждения часто называются теоремами, леммами, определениями, постулатами, замечаниями, выводами, алгоритмами, фактами, предположениями или некоторыми подобными вещами, и они обычно обрабатываются специальным образом. Макрокоманда `\proclaim`, которая иллюстрировалась в этом приложении ранее, а также в главе 20, помещает заголовок объявления жирным шрифтом, затем устанавливает остаток абзаца в наклонный шрифт. За абзацем следует что-то похожее на `\medbreak`, за тем исключением, что величина штрафа другая, так что разрыв страницы не одобряется:

```
\outer\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2}\par
  \ifdim\lastskip<\medskipamount \removelastskip\penalty55\medskip\fi}
```

Неровный правый край инициализируется тем, что промежутки между словами имеют фиксированную ширину и что справа от каждой строки помещаются промежутки различной ширины. Вы не должны вызывать `\raggedright` до тех пор, пока не задан текстовой шрифт. Предполагается, что материал с неровным правым краем не будет набираться в различных размерах. (Если это не так, должен быть использован другой подход: параметры 3 и 4 `\fontdimen` тех шрифтов, которые вы будете использовать, должны быть установлены в ноль, например, командой `\fontdimen3\tenrm=0pt`. Эти параметры задают растяжимость и сжимаемость промежутков между словами.) Для установления неровного правого края в шрифте пишущей машинки должна быть использована специальная макрокоманда `\ttraggedright`, поскольку в этом шрифте промежутки между словами обычно больше. (В шрифте `cmtt` промежутки уже нерастяжимы и несжимаемы.)

```
\def\raggedright{\rightskip=0pt plus2em
  \spaceskip=.3333em \xspaceskip=.5em }
\def\ttraggedright{\tt\rightskip=0pt plus2em }
```

Теперь мы подошли к специальным символам и акцентам, которые основаны главным образом на символах, имеющихся в Computer Modern шрифтах. Если используются другие стили печати, потребуются другие конструкции. Когда при помощи формирования бокса строится символ, сначала вызывается макрокоманда `\leavevmode`. Это начинает новый абзац, если TEX находится в вертикальной моде, и не делает ничего, если TEX в горизонтальной или математической моде.

```
\chardef\%='\% \chardef\&='\& \chardef\#='\# \chardef\$='\$
\chardef\ss="19
\chardef\ae="1A \chardef\oe="1B \chardef\o="1C
\chardef\AE="1D \chardef\OE="1E \chardef\O="1F
\chardef\i="10 \chardef\j="11 % dotless letters
\def\aa{\accent'27a} \def\l{\char'40l}

\def\leavevmode{\unhbox\voidb@x} % begins a paragraph, if necessary
\def\_{\leavevmode \kern.06em \vbox{\hrule width0.3em}}
\def\L{\leavevmode\setbox0=\hbox{L}\hbox to\wd0{\hss\char'40L}}
\def\AA{\leavevmode\setbox0=\hbox{h}\dimen@=\ht0 \advance\dimen@ by-1ex
  \rlap{\raise.67\dimen@\hbox{\char'27}}A}
```

```
\def\mathhexbox#1#2#3{\leavevmode
  \hbox{$\m@th \mathchar"#1#2#3$}}
\def\dag{\mathhexbox279}  \def\ddag{\mathhexbox27A}
\def\S{\mathhexbox278} \def\P{\mathhexbox27B}

\def\oalign#1{\leavevmode\vtop{\baselineskip0pt \lineskip.25ex
  \ialign{##\crcr#1\crcr}}} % put characters over each other
\def\ooalign{\lineskiplimit-\maxdimen \oalign}
\def\d#1{\oalign{#1\crcr\hidewidth.\hidewidth}}
\def\b#1{\oalign{#1\crcr\hidewidth
    \vbox to.2ex{\hbox{\char'22}\vss}\hidewidth}}
\def\c#1{\setbox0=\hbox{#1}\ifdim\ht0=1ex \accent'30 #1%
  \else{\ooalign{\hidewidth\char'30\hidewidth\crcr\unhbox0}}\fi}
\def\copyright{{\ooalign
    {\hfil\raise.07ex\hbox{c}\hfil\crcr\mathhexbox20D}}}

\def\dots{\relax\ifmmode\ldots\else$\m@th \ldots\,$\fi}
\def\TeX{T\kern-.1667em \lower.5ex\hbox{E}\kern-.125em X}

\def\`#1{{\accent"12 #1}} \def\'#1{{\accent"13 #1}}
\def\v#1{{\accent"14 #1}} \def\u#1{{\accent"15 #1}}
\def\=#1{{\accent"16 #1}} \def\^#1{{\accent"5E #1}}
\def\.#1{{\accent"5F #1}} \def\H#1{{\accent"7D #1}}
\def\~#1{{\accent"7E #1}} \def\"#1{{\accent"7F #1}}
\def\t#1{{\edef\next{\the\font}\the\textfont1\accent"7F\next#1}}
```

В этом месте определяются три альтернативных символа акцента, которые подходят для клавиатур с расширенным набором символов (ср. приложение C):

```
\let\^^_=\v       \let\^^S=\u       \let\^^D=\^
```

Далее предусмотрены различные способы заполнения пространства проводниками.

```
\def\hrulefill{\leaders\hrule\hfill}
\def\dotfill{\cleaders\hbox{$\m@th \mkern1.5mu . \mkern1.5mu$}\hfill}
\def\rightarrowfill{$\m@th \mathord- \mkern-6mu
  \cleaders\hbox{$\mkern-2mu \mathord- \mkern-2mu$}\hfill
  \mkern-6mu \mathord\rightarrow$}
\def\leftarrowfill{$\m@th \mathord\leftarrow \mkern-6mu
  \cleaders\hbox{$\mkern-2mu \mathord- \mkern-2mu$}\hfill
  \mkern-6mu \mathord-$}

\mathchardef\braceld="37A \mathchardef\bracerd="37B
\mathchardef\braceru="37C \mathchardef\braceru="37D
\def\upbracefill{$\m@th
  \braceru\leaders\vrule\hfill\bracerd
  \braceld\leaders\vrule\hfill\braceru$}
\def\downbracefill{$\m@th
  \braceld\leaders\vrule\hfill\braceru
  \braceru\leaders\vrule\hfill\bracerd$}
```

Макрокоманды `\upbracefill` и `\downbracefill` имеют ограниченное использование: они должны появляться *сами по себе* в горизонтальном боксе или в элементе выравнивания, за исключением горизонтальных пробелов.

И, наконец, завершается пятая секция `plain.tex` определением `\bye`:

```
\outer\def\bye{\par\vfill\supereject\end} % the recommended way to stop
```

6. Математические макрокоманды. Шестой раздел `plain.tex` самый длинный, но здесь достаточно привести только выдержки из него, поскольку большая его часть — это просто скучный перечень специальных символов с их расположением в шрифтах, а та же самая информация приведена в приложении F.

Сначала идут некоторые элементарные вещи. Для тех, кто не может просто вводить `^` и `_`, есть команды `\sp` и `\sb`, приводятся четыре команды для коррекции пробелов, определяется "возможный знак умножения" `\*`, затем следует интересный набор макрокоманд, который превращает `f'''` в `f^{\prime\prime\prime}`:

```
\let\sp=^ \let\sb=_
\def\,{\mskip\thinmuskip}  \def\!{\mskip-\thinmuskip}
\def\>{\mskip\medmuskip}   \def\;{\mskip\thickmuskip}
\def\*{\discretionary{\thinspace\the\textfont2\char2}{}{}}
{\catcode`\^^Z=\active \gdef^^Z{\not=}} % ^^Z is like \ne in math
{\catcode`\'=\active \gdef'{^\bgroup\prim@s}}
\def\prim@s{\prime\futurelet\next\pr@m@s}
\def\pr@m@s{\ifx'\next\let\next\pr@@@s \else\ifx^\next\let\next\pr@@@t
  \else\let\next\egroup\fi\fi \next}
\def\pr@@@s#1{\prim@s} \def\pr@@@t#1#2{#2\egroup}
```

Следующая работа — определить греческие буквы и другие символы типа Ord. Прописным греческим буквам присваиваются шестнадцатиричные коды вида `"7xxx`, так что при изменении `\fam` они будут изменять семейства. Здесь и ниже многоточия "$\cdots$" используются для того, чтобы указать, что в приложении F перечислены и другие символы, имеющие аналогичное определение.

```
\mathchardef\alpha="010B    ...    \mathchardef\omega="0121
\mathchardef\Gamma="7000    ...    \mathchardef\Omega="700A
\mathchardef\aleph="0240    ...    \mathchardef\spadesuit="027F
\def\hbar{{\mathchar'26\mkern-9muh}}
\def\surd{{\mathchar"1270}}
\def\angle{{\vbox{\ialign{$\m@th\scriptstyle##$\crcr
      \not\mathrel{\mkern14mu}\crcr \noalign{\nointerlineskip}
      \mkern2.5mu\leaders\hrule height.34pt\hfill\mkern2.5mu\crcr}}}}
```

Далее большим операторам присваиваются шестнадцатиричные коды типа `"1xxx`:

```
\mathchardef\smallint="1273
\mathchardef\sum="1350    ...    \mathchardef\biguplus="1355
\mathchardef\intop="1352  \def\int{\intop\nolimits}
\mathchardef\ointop="1348 \def\oint{\ointop\nolimits}
```

Знаки интегралов обрабатываются специально, так что пределы интегрирования не задаются над и под интегралами.

Далее идут бинарные операции, здесь нет ничего волнующего.

```
\mathchardef\pm="2206    ...    \mathchardef\amalg="2271
```

Отношения тоже совершенно прямолинейны, за исключением тех, которые сконструированы из других символов. `\mapstochar` является символом ↦ нулевой ширины, который сам по себе совершенно бесполезен, но комбинируется с правыми стрелками, чтобы сделать `\mapsto` "↦" и `\longmapsto` "⟼". Аналогично, `\not` — это символ отношения нулевой ширины, который помещает слэш на следующим за ним символе. Когда в математической формуле соседствуют два отношения, TEX не помещает между ними пробела.

```
\mathchardef\leq="3214    ...    \mathchardef\perp="323F
\def\joinrel{\mathrel{\mkern-3mu}}
\def\relbar{\mathrel{\smash-}} \def\Relbar{\mathrel=}
\def\longrightarrow{\relbar\joinrel\rightarrow}
\def\Longrightarrow{\Relbar\joinrel\Rightarrow}
\def\longleftarrow{\leftarrow\joinrel\relbar}
\def\Longleftarrow{\Leftarrow\joinrel\Relbar}
\def\longleftrightarrow{\leftarrow\joinrel\rightarrow}
\def\Longleftrightarrow{\Leftarrow\joinrel\Rightarrow}
\mathchardef\mapstochar="322F \def\mapsto{\mapstochar\rightarrow}
\def\longmapsto{\mapstochar\longrightarrow}
\mathchardef\lhook="312C \def\hookrightarrow{\lhook\joinrel\rightarrow}
\mathchardef\rhook="312D \def\hookleftarrow{\leftarrow\joinrel\rhook}

\def\neq{\not=}   \def\models{\mathrel|\joinrel=}
\def\bowtie{\mathrel\triangleright\joinrel\mathrel\triangleleft}
```

После определения символов `\ldotp` и `\cdotp`, действующих как математическая пунктуация, легко определить макрокоманды `\ldots` и `\cdots`, которые в большинстве случаев дают подходящие пробелы Также предусмотрены вертикальные и диагональные многоточия (`\vdots` и `\ddots`):

```
\mathchardef\ldotp="602E\mathchardef\cdotp="6201\mathchardef\colon="603A
\def\ldots{\mathinner{\ldotp\ldotp\ldotp}}
\def\cdots{\mathinner{\cdotp\cdotp\cdotp}}
\def\vdots{\vbox{\baselineskip=4pt \lineskiplimit=0pt
    \kern6pt \hbox{.}\hbox{.}\hbox{.}}}
\def\ddots{\mathinner{\mkern1mu\raise7pt\vbox{\kern7pt\hbox{.}}\mkern2mu
    \raise4pt\hbox{.}\mkern2mu\raise1pt\hbox{.}\mkern1mu}}
```

С большинством математических акцентов прекрасно справляются при помощи примитива `\mathaccent`, но некоторые из них, имеющие переменную ширину, создаются более трудным способом:

```
\def\acute{\mathaccent"7013 }    ...    \def\ddot{\mathaccent"707F }
\def\widetilde{\mathaccent"0365 } \def\widehat{\mathaccent"0362 }
```

```
\def\overrightarrow#1{\vbox{\ialign{##\crcr
     \rightarrowfill\crcr\noalign{\kern-1pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overleftarrow#1{\vbox{\ialign{##\crcr
     \leftarrowfill\crcr\noalign{\kern-1pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overbrace#1{\mathop{\vbox{\ialign{##\crcr\noalign{\kern3pt}
     \downbracefill\crcr\noalign{\kern3pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}\limits}
\def\underbrace#1{\mathop{\vtop{\ialign{##\crcr
     $\hfil\displaystyle{#1}\hfil$\crcr\noalign{\kern3pt\nointerlineskip}
     \upbracefill\crcr\noalign{\kern3pt}}}}\limits}
\def\skew#1#2#3{{#2{#3\mkern#1mu}\mkern-#1mu}{}}
```

Теперь мы подошли к 24 ограничителям, которые могут изменять свой размер:

```
\def\langle{\delimiter"426830A }      \def\rangle{\delimiter"526930B }
\def\lbrace{\delimiter"4266308 }      \def\rbrace{\delimiter"5267309 }
\def\lceil{\delimiter"4264306 }       \def\rceil{\delimiter"5265307 }
\def\lfloor{\delimiter"4262304 }      \def\rfloor{\delimiter"5263305 }
\def\lgroup{\delimiter"400033A }      \def\rgroup{\delimiter"500033B }
\def\lmoustache{\delimiter"4000340 }  \def\rmoustache{\delimiter"5000341 }
\def\uparrow{\delimiter"3222378 }     \def\Uparrow{\delimiter"322A37E }
\def\downarrow{\delimiter"3223379 }   \def\Downarrow{\delimiter"322B37F }
\def\updownarrow{\delimiter"326C33F } \def\arrowvert{\delimiter"000033C }
\def\Updownarrow{\delimiter"326D377 } \def\Arrowvert{\delimiter"000033D }
\def\vert{\delimiter"026A30C }        \def\Vert{\delimiter"026B30D }
\def\backslash{\delimiter"026E30F }   \def\bracevert{\delimiter"000033E }
```

Конструкция `\big...\Bigg` производит специфические размеры:

```
\def\bigl{\mathopen\big} \def\bigm{\mathrel\big} \def\bigr{\mathclose\big}
\def\Bigl{\mathopen\Big} \def\Bigm{\mathrel\Big} \def\Bigr{\mathclose\Big}
\def\biggl{\mathopen\bigg}            \def\Biggl{\mathopen\Bigg}
\def\biggm{\mathrel\bigg}             \def\Biggm{\mathrel\Bigg}
\def\biggr{\mathclose\bigg}           \def\Biggr{\mathclose\Bigg}
\def\big#1{{\hbox{$\left#1\vbox to 8.5pt{}\right.\n@space$}}}
\def\Big#1{{\hbox{$\left#1\vbox to 11.5pt{}\right.\n@space$}}}
\def\bigg#1{{\hbox{$\left#1\vbox to 14.5pt{}\right.\n@space$}}}
\def\Bigg#1{{\hbox{$\left#1\vbox to 17.5pt{}\right.\n@space$}}}
\def\n@space{\nulldelimiterspace=0pt \m@th}
```

Есть несколько других аббревиатур, относящихся к ограничителям:

```
\def\choose{\atopwithdelims()}
\def\brack{\atopwithdelims[]}
\def\brace{\atopwithdelims\{\}}
\def\sqrt{\radical"270370 }
```

А теперь будет нечто более интересное. Операция `\mathpalette` создает формулы во всех четырех стилях. Здесь она применяется в реализации `\phantom`, `\smash`, `\root` и других операций.

```
\def\mathpalette#1#2{\mathchoice{#1\displaystyle{#2}}
  {#1\textstyle{#2}}{#1\scriptstyle{#2}}{#1\scriptscriptstyle{#2}}}

\newbox\rootbox
\def\root#1\of{\setbox\rootbox=
  \hbox{$\m@th \scriptscriptstyle{#1}$}
  \mathpalette\r@@t}
\def\r@@t#1#2{\setbox0=\hbox{$\m@th #1\sqrt{#2}$}
  \dimen@=\ht0 \advance\dimen@ by-\dp0
  \mkern5mu \raise.6\dimen@\copy\rootbox \mkern-10mu \box0}

\newif\ifv@ \newif\ifh@
\def\vphantom{\v@true\h@false\ph@nt}
\def\hphantom{\v@false\h@true\ph@nt}
\def\phantom{\v@true\h@true\ph@nt}
\def\ph@nt{\ifmmode\def\next{\mathpalette\mathph@nt}%
  \else\let\next=\makeph@nt\fi \next}
\def\makeph@nt#1{\setbox0=\hbox{#1}\finph@nt}
\def\mathph@nt#1#2{\setbox0=\hbox{$\m@th#1{#2}$}\finph@nt}
\def\finph@nt{\setbox2=\null \ifv@ \ht2=\ht0 \dp2=\dp0 \fi
  \ifh@ \wd2=\wd0 \fi \box2 }
\def\mathstrut{\vphantom(}

\def\smash{\relax % \relax, in case this comes first in \halign
  \ifmmode\def\next{\mathpalette\mathsm@sh}\else\let\next\makesm@sh
  \fi \next}
\def\makesm@sh#1{\setbox0=\hbox{#1}\finsm@sh}
\def\mathsm@sh#1#2{\setbox0=\hbox{$\m@th#1{#2}$}\finsm@sh}
\def\finsm@sh{\ht0=0pt \dp0=0pt \box0 }

\def\cong{\mathrel{\mathpalette\@vereq\sim}} % \sim over =
\def\@vereq#1#2{\lower.5pt\vbox{\baselineskip0pt \lineskip-.5pt
    \ialign{$\m@th#1\hfil##\hfil$\crcr#2\crcr=\crcr}}}
\def\notin{\mathrel{\mathpalette\c@ncel\in}}
\def\c@ncel#1#2{\ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
\def\rightleftharpoons{\mathrel{\mathpalette\rlh@{}}}
\def\rlh@#1{\vcenter{\hbox{\ooalign{\raise2pt
          \hbox{$#1\rightharpoonup$}\crcr $#1\leftharpoondown$}}}}
\def\buildrel#1\over#2{\mathrel{\mathop{\kern0pt #2}\limits^{#1}}}
\def\doteq{\buildrel\textstyle.\over=}
```

Эти определения показывают, как можно определить другие составные комбинации символов, чтобы они работали во всех четырех стилях.

Теперь определяются альтернативные имена:

```
\let\ne=\neq        \let\le=\leq         \let\ge=\geq
\let\{=\lbrace      \let\|=\Vert         \let\}=\rbrace
\let\to=\rightarrow \let\gets=\leftarrow \let\owns=\ni
```

```
\let\land=\wedge    \let\lor=\vee        \let\lnot=\neg
\def\iff{\;\Longleftrightarrow\;}
```

В главе 18 перечислены 32 функции, имена которых печатаются прямым шрифтом. Здесь следует показать только несколько из них:

```
\def\arccos{\mathop{\rm arccos}\nolimits}
                    ...    \def\tanh{\mathop{\rm tanh}\nolimits}
\def\det{\mathop{\rm det}}    ...    \def\sup{\mathop{\rm sup}}
\def\liminf{\mathop{\rm lim\,inf}} \def\limsup{\mathop{\rm lim\,sup}}

\def\bmod{\mskip-\medmuskip \mkern5mu
  \mathbin{\rm mod} \penalty900 \mkern5mu \mskip-\medmuskip}
\def\pmod#1{\allowbreak \mkern18mu ({\rm mod}\,\,#1)}
```

Определение `\matrix` требует некоторых усилий, чтобы гарантировать, что две n-строчные матрицы будут иметь одинаковые ширину и глубину, если только какой-нибудь их ряд не слишком большой. Определение `\bordermatrix` еще более сложное, но оказывается, оно достаточно хорошо работает. Это определение использует константу `\p@renwd`, которая равна ширине большой растяжимой круглой скобки.

```
\def\matrix#1{\null\,\vcenter{\normalbaselines\m@th
    \ialign{\hfil$##$\hfil&&\quad\hfil$##$\hfil\crcr
      \mathstrut\crcr\noalign{\kern-\baselineskip}
      #1\crcr\mathstrut\crcr\noalign{\kern-\baselineskip}}}\,}

\newdimen\p@renwd \setbox0=\hbox{\tenex B} \p@renwd=\wd0
\def\bordermatrix#1{\begingroup \m@th
  \setbox0=\vbox{\def\cr{\crcr\noalign{\kern2pt\global\let\cr=\endline}}
      \ialign{$##$\hfil\kern2pt\kern\p@renwd&\thinspace\hfil$##$\hfil
        &&\quad\hfil$##$\hfil\crcr
        \omit\strut\hfil\crcr\noalign{\kern-\baselineskip}
        #1\crcr\omit\strut\cr}}
  \setbox2=\vbox{\unvcopy0 \global\setbox1=\lastbox}
  \setbox2=\hbox{\unhbox1 \unskip \global\setbox1=\lastbox}
  \setbox2=\hbox{$\kern\wd1\kern-\p@renwd \left( \kern-\wd1
    \global\setbox1=\vbox{\box1\kern2pt}
    \vcenter{\kern-\ht1 \unvbox0 \kern-\baselineskip} \,\right)$}
  \null\;\vbox{\kern\ht1\box2}\endgroup}
```

Следующие макрокоманды намного проще:

```
\def\cases#1{\left\{\,\vcenter{\normalbaselines\m@th
    \ialign{$##\hfil$&\quad##\hfil\crcr#1\crcr}}\right.}
\def\pmatrix#1{\left( \matrix{#1} \right)}
```

И наконец, макрокоманды для выделенных уравнений:

```
\def\openup{\afterassignment\@penup\dimen@=}
\def\@penup{\advance\lineskip\dimen@
  \advance\baselineskip\dimen@ \advance\lineskiplimit\dimen@}
```

```
\def\eqalign#1{\null\,\vcenter{\openup1\jot \m@th
  \ialign{\strut\hfil$\displaystyle{##}$&$\displaystyle{{}##}$\hfil
      \crcr#1\crcr}}\,}
\newif\ifdt@p
\def\displ@y{\global\dt@ptrue \openup1\jot \m@th
  \everycr{\noalign{\ifdt@p \global\dt@pfalse
      \vskip-\lineskiplimit \vskip\normallineskiplimit
      \else \penalty\interdisplaylinepenalty \fi}}
\def\@lign{\tabskip=0pt\everycr={}} % restore inside \displ@y
\def\displaylines#1{\displ@y
  \halign{\hbox to\displaywidth{$\hfil\@lign\displaystyle##\hfil$}\crcr
    #1\crcr}}
\def\eqalignno#1{\displ@y \tabskip=\centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip=0pt
    &$\@lign\displaystyle{{}##}$\hfil\tabskip=\centering
    &\llap{$\@lign##$}\tabskip=0pt\crcr
    #1\crcr}}
\def\leqalignno#1{\displ@y \tabskip=\centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip=0pt
    &$\@lign\displaystyle{{}##}$\hfil\tabskip=\centering
    &\kern-\displaywidth\rlap{$\@lign##$}\tabskip=\displaywidth\crcr
    #1\crcr}}
```

Предполагается, что `\lineskiplimit` — это `\normallineskiplimit` плюс накопленная величина "открытия". Таким образом, инструкция `\vskip` в `\displ@y` будет компенсировать тот факт, что первая базовая линия выравнивания отделена открывающей базовой линией от последней строки того, что предшествует выделению.

7. Макрокоманды для вывода. Файл `plain.tex` также содержит программу вывода, описанную в главах 15 и 23. Сначала несколько простых возможностей, относящихся к номерам страниц, оформлению верхней и нижней строки страницы:

```
\countdef\pageno=0 \pageno=1 % first page is number 1
\newtoks\headline \headline={\hfil} % headline is normally blank
\newtoks\footline \footline={\hss\tenrm\folio\hss}
  % footline is normally a centered page number in font \tenrm
\def\folio{\ifnum\pageno<0 \romannumeral-\pageno \else\number\pageno \fi}
\def\nopagenumbers{\footline={\hfil}} % blank out the footline
\def\advancepageno{\ifnum\pageno<0 \global\advance\pageno by -1
  \else\global\advance\pageno by 1 \fi} % increase |pageno|
\newif\ifr@ggedbottom
\def\raggedbottom{\topskip10pt plus60pt \r@ggedbottomtrue}
\def\normalbottom{\topskip10pt \r@ggedbottomfalse} % undoes \raggedbottom
```

Макрокоманда `\footnote` имеет несколько тонких особенностей, которые могут оценить те, кто очень внимательно читал главу 15. Она также использует

некоторые хитрости \bgroup, \futurelet и \aftergroup, так что текст сноски не должен быть параметром \vfootnote:

```
\newinsert\footins
\def\footnote#1{\let\@sf=\empty % parametre #2 (the text) is read later
  \ifhmode\edef\@sf{\spacefactor=\the\spacefactor}\/\fi
  #1\@sf\vfootnote{#1}}
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty=\interfootnotelinepenalty
  \splittopskip=\ht\strutbox % top baseline for broken footnotes
  \splitmaxdepth=\dp\strutbox \floatingpenalty=20000
  \leftskip=0pt \rightskip=0pt \spaceskip=0pt \xspaceskip=0pt
  \textindent{#1}\footstrut\futurelet\next\fo@t}
\def\fo@t{\ifcat\bgroup\noexpand\next \let\next\f@@t
  \else\let\next\f@t\fi \next}
\def\f@@t{\bgroup\aftergroup\@foot\let\next}
\def\f@t#1{#1\@foot}
\def\@foot{\strut\egroup}
\def\footstrut{\vbox to\splittopskip{}}
\skip\footins=\bigskipamount % space added when footnote is present
\count\footins=1000 % footnote magnification factor (1 to 1)
\dimen\footins=8in % maximum footnotes per page
```

Плавающими вставками управляет команда \insert, вертикальный список которой состоит из штрафа со следующим за ним одним боксом:

```
\newinsert\topins \newif\ifp@ge \newif\if@mid
\def\topinsert{\@midfalse\p@gefalse\@ins}
\def\midinsert{\@midtrue\@ins}
\def\pageinsert{\@midfalse\p@getrue\@ins}
\skip\topins=0pt % no space added when a topinsert is present
\count\topins=1000 % magnification factor (1 to 1)
\dimen\topins=\maxdimen % no limit per page
\def\@ins{\par\begingroup\setbox0=\vbox\bgroup} % start a \vbox
\def\endinsert{\egroup % finish the \vbox
  \if@mid \dimen@=\ht0 \advance\dimen@ by\dp0
    \advance\dimen@ by12\p@ \advance\dimen@ by\pagetotal
    \ifdim\dimen@>\pagegoal \@midfalse\p@gefalse\fi\fi
  \if@mid \bigskip \box0 \bigbreak
  \else\insert\topins{\penalty100 % floating insertion
    \splittopskip=0pt \splitmaxdepth=\maxdimen \floatingpenalty=0
    \ifp@ge \dimen@=\dp0
     \vbox to\vsize{\unvbox0 \kern-\dimen@} % depth is zero
    \else \box0 \nobreak\bigskip\fi}\fi\endgroup}
```

Большая часть программы \output приведена в главе 23. Здесь она дана полностью:

```
\output={\plainoutput}
```

```
\def\plainoutput{\shipout\vbox{\makeheadline\pagebody\makefootline}%
  \advancepageno
  \ifnum\outputpenalty>-20000 \else\dosupereject\fi}
\def\pagebody{\vbox to\vsize{\boxmaxdepth=\maxdepth \pagecontents}}
\def\makeheadline{\vbox to 0pt{\vskip-22.5pt
    \line{\vbox to8.5pt{}\the\headline}\vss}\nointerlineskip}
\def\makefootline{\baselineskip=24pt \line{\the\footline}}
\def\dosupereject{\ifnum\insertpenalties>0 % something is being held over
  \line{}\kern-\topskip\nobreak\vfill\supereject\fi}

\def\pagecontents{\ifvoid\topins\else\unvbox\topins\fi
  \dimen@=\dp255 \unvbox255
  \ifvoid\footins\else % footnote info is present
    \vskip\skip\footins \footnoterule \unvbox\footins\fi
  \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
\def\footnoterule{\kern-3pt
  \hrule width 2truein \kern 2.6pt} % the \hrule is .4pt high
```

8. Перенос и все остальное. Последняя часть `plain.tex` читает образцы переноса и исключений, найденные в файле `hyphen.tex` (см. приложение H). Затем определяется несколько разнообразных команд, устанавливается шрифт `\rm`, и это все!

```
\input hyphen % the hyphenation patterns and exceptions

\def\magnification{\afterassignment\m@g\count@}
\def\m@g{\mag=\count@
  \hsize6.5truein\vsize8.9truein\dimen\footins8truein}

\def\tracingall{\tracingonline=1 \tracingcommands=2 \tracingstats=2
  \tracingpages=1 \tracingoutput=1 \tracinglostchars=1
  \tracingmacros=2 \tracingparagraphs=1 \tracingrestores=1
  \showboxbreadth=\maxdimen \showboxdepth=\maxdimen \errorstopmode}

\def\showhyphens#1{\setbox0=\vbox{\parfillskip0pt \hsize=\maxdimen \tenrm
  \pretolerance=-1 \tolerance=-1 \hbadness=0 \showboxdepth=0 \ #1}}

\normalbaselines\rm % select roman font
\nonfrenchspacing % punctuation affects the spacing
\catcode`@=12 %  at signs are no longer letters

\def\fmtname{plain}\def\fmtversion{2.0} % identifies the current format
```

Имя формата и номер версии записывается в команде для того, чтобы помочь тем, кто должен объяснить, почему что-то не работает. Макрофайлы типа `plain.tex` не должны изменяться, разве что по отношению к предварительно загруженным шрифтам, если только изменения не вносятся автором формата.

Назначение системы программирования — сделать компьютер удобным для использования. Для этого система предлагает языки и различные услуги, которые фактически есть программы, вызываемые и управляемые языковыми средствами. Но эти удобства даются не даром: Документация к системе программирования в десять – двадцать раз больше, чем документация к самому компьютеру. Программисту становится много легче выполнить любую нужную функцию, но приходится выбирать ее из намного более обширного набора и помнить много больше вариантов и форматов.

The purpose of a programming system is to make a computer easy to use. To do this, it furnishes languages and various facilities that are in fact programs invoked and controlled by language features. But these facilities are bought at a price: the external description of a programming system is ten to twenty times as large as the external description of the computer system itself. The user finds it far easier to specify any particular function, but there are far more to choose from, and far more options and formats to remember.

—FREDERICK P. BROOKS, JR., *The Mythical Man Month* (1975)

Если кто-нибудь скажет: "Мне нужен такой язык программирования, в котором мне надо только сказать, что я хочу, чтобы было сделано," — дайте ему леденец.

When someone says, "I want a programming language in which I need only say what I wish done," give him a lollipop.

—ALAN PERLIS, *Epigrams on Programming* (1982)

C

Символьные коды

На разных компьютерах символы представлены по-разному, но результаты TEX'а одинаковы на всех машинах, так как при чтении он все преобразует в стандартный внутренний код. TEX также преобразует внутреннее представление во внешний код, когда записывает файл, поэтому обычно не нужно знать, как в машине переключаются коды.

Это приложение описывает внутренний код TEX'а, одинаковый для всех реализаций. Наличие такого кода весьма существенно, так как делает конструкции TEX'а мобильными. Например, можно использовать буквенные константы типа `‘b` как числа. То, что `‘b` означает целое 98, значит, что можно написать машинно-независимые макросы, которые, например, решают, является ли данный символ цифрой от `0` до `9`. Внутренняя кодировка TEX'а также работает в `dvi`-файлах, которые можно напечатать программой, не знающей, где подготовлены `dvi`-данные. Все реализации TEX'а дают одинаковые результаты, независимо от основной ЭВМ, потому что `dvi`-данные выражены в машинно-независимом коде.

Внутренний код TEX'а основан на Американском стандартном коде для обмена информацией, который обычно называют “ASCII”. Имеются 128 кодов с номерами от 0 до 127. По традиции, числа выражаются в восьмеричной системе от *′000* до *′177* или в шестнадцатиричной системе от `"00` до `"7F`. Таким образом, `‘b` обычно называется *′142* или `"62`, а не 98. В ASCII коды от *′000* до *′040* и код *′177* присвоены специальным функциям. Например, код *′007* называется `BEL` (“звонок”). Остальные 94 кода присвоены видимым символам. Приведем схему, которая показывает коды ASCII так, что можно легко прочитать восьмеричные и шестнадцатиричные эквиваленты:

	′0	*′1*	*′2*	*′3*	*′4*	*′5*	*′6*	*′7*	
′00x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	`"0x`
′01x	BS	HT	LF	VT	FF	CR	SO	SI	
′02x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	`"1x`
′03x	CAN	EM	SUB	ESC	FS	GS	RS	US	
′04x	SP	!	"	#	$	%	&	’	`"2x`
′05x	(	)	*	+	,	-	.	/	
′06x	0	1	2	3	4	5	6	7	`"3x`
′07x	8	9	:	;	<	=	>	?	
′10x	@	A	B	C	D	E	F	G	`"4x`
′11x	H	I	J	K	L	M	N	O	
′12x	P	Q	R	S	T	U	V	W	`"5x`
′13x	X	Y	Z	[	\	]	^	_	
′14x	‘	a	b	c	d	e	f	g	`"6x`
′15x	h	i	j	k	l	m	n	o	
′16x	p	q	r	s	t	u	v	w	`"7x`
′17x	x	y	z	{	|	}	~	DEL	
	`"8`	`"9`	`"A`	`"B`	`"C`	`"D`	`"E`	`"F`	

С тех пор, как в начале 60-х годов был введен код ASCII, у многих возникали различные идеи насчет того, что делать с позициями *'000–'037* и *'177*, потому что большинство функций, присвоенных этим кодам, подходит только для специальных целей, таких как передача файлов, а не для приложений, связанных с печатью или интерактивными вычислениями. Фирмы-изготовители вскоре начали производить строчные принтеры, которые могли генерировать 128 символов, 33 из которых были предназначены для специальных нужд конкретных заказчиков, таким образом, преимущества стандартного кода частично терялись. С другой стороны, оставшиеся 95 кодов (включая *'40*=SP, пробел) были общеприняты, и сейчас они внедрены на большинстве современных компьютерных терминалах. Имея клавиатуру ASCII, можно задать каждый из 128 кодов TEX'а в терминах 95 стандартных символов следующим образом:

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*	
'00x	^^@	^^A	^^B	^^C	^^D	^^E	^^F	^^G	"0x
'01x	^^H	^^I	^^J	^^K	^^L	^^M	^^N	^^O	
'02x	^^P	^^Q	^^R	^^S	^^T	^^U	^^V	^^W	"1x
'03x	^^X	^^Y	^^Z	^^[	^^\	^^]	^^^	^^_	
'04x	␣	!	"	#	$	%	&	'	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	`	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{	|	}	~	^^?	
	"8	"9	"A	"B	"C	"D	"E	"F	

(Здесь ^^ не обязательно означает два символа сиркомфлекса: это означает два идентичных символа, у которых текущий \catcode равен 7. В таких случаях TEX просто прибавляет или вычитает *'100* из внутреннего кода следующего за ним символа. Например, * можно ввести как ^^j, а j можно также ввести как ^^*.)

Примерно в 1965 году в нескольких университетах был разработан расширенный код ASCII, предназначенный для редактирования текстов и интерактивных вычислений, и в течение нескольких лет в Стэнфорде, MIT, Карнеги-Меллоне и других местах использовались терминалы, имеющие не 95, а 120 или 121 символ. Любители таких клавиатур (в том числе и автор этой книги) неохотно отказываются от лишних символов. Похоже, что такие люди интенсивно используют примерно 5 из лишних 25 символов, а оставшиеся 20 используют лишь время от

времени, хотя у разных людей эти пятерки символов будут разными. Например, автор разработал TEX на клавиатуре, которая включает символы $\leftarrow$, $\downarrow$, $\neq$, $\leq$ и $\geq$, и ему кажется, что с их помощью приятнее печатать учебные заметки, технические бумаги и машинные программы, которые он любит писать. Его друзья-логики интенсивно используют клавиши $\forall$, $\exists$ и т. д. Рекомендуется, чтобы реализации TEX на системах с большими наборами символов содержали следующие коды:

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*	
'00x	$\cdot$	$\downarrow$	α	β	$\wedge$	$\neg$	$\in$	π	"0x
'01x	λ	γ	δ	$\uparrow$	$\pm$	$\oplus$	ϖ	∂	
'02x	$\subset$	$\supset$	$\cap$	$\cup$	$\forall$	$\exists$	$\otimes$	$\leftrightarrows$	"1x
'03x	$\leftarrow$	$\rightarrow$	$\neq$	$\diamond$	$\leq$	$\geq$	$\equiv$	$\vee$	
'04x		!	"	#	$	%	&	’	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	‘	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{	|	}	~	$\int$	
	"8	"9	"A	"B	"C	"D	"E	"F	

Конечно, разработчикам пакетов макрокоманд, которые предназначены для широкого применения, следует придерживаться стандартных символов ASCII.

Между прочим, символ ASCII ^ , который находится в позиции *'136*, иногда называется “caret”, но английские словари указывают, что на самом деле caret (полиграфический знак вставки) — больший по размеру символ, он скорее похож на символ *'004* в приведенном выше расширенном наборе символов. Правильное название ^ — “сиркомфлекс”, но это труднопроизносимое слово, так что предпочтительнее называть его короче и проще - “шляпка”(“hat”). Желательно сохранить традиционное различие между знаком вставки (caret) и шляпкой.

Приведенный выше расширенный набор символов разработан в MIT. Он очень похож на код, который сейчас используется в Стэнфорде, но немного лучше его. Семь кодов традиционно присваиваются стандартным управляющим функциям ASCII: `NUL` (⟨null⟩), `HT` (⟨tab⟩), `LF` (⟨linefeed⟩), `FF` (⟨formfeed⟩), `CR` (⟨return⟩), `ESC` (⟨escape⟩), and `DEL` (⟨delete⟩). Они находятся в стандартных позициях ASCII, следовательно, соответствующие семь символов $\cdot$ γ δ $\pm$ $\oplus$ $\diamond$ $\int$ в действительности не появляются на клавиатуре. Семь “скрытых” символов появляются только

на определенных выходных устройствах.

Современные клавматуры позволяют вводить не 125, а 256 символов, поэтому TEX внутренне представляет символы как числа в пределах 0 – 255 (то есть '000 – '377 или "00 – "FF). Реализации TEX'а различаются по тому, какие символы разрешены во входных файлах и во что они преобразуются в выходных файлах. Эти подмножества можно задавать независимо. Полная допучтимая версия TEX'а позволяет иметь все 256 символов и на входе, и на выходе, другие версии могут игнорировать все, кроме видимых символов ASCII. Могут существовать версии, которые на входе отличают символ табуляции (код '011) от пробела и выводят его как последовательность трех символов `^^I`.

К сожалению, многие сталкиваются с противоположной проблемой: вместо 95 стандартных и нескольких других символов у них в действительности менее 95 символов. Что можно делать в таких случаях? Можно, скажем, использовать TEX с меньшим количеством символов, используя больше команд, например, начальный TEX определяет `\lq`, `\rq`, `\lbrack`, `\rbrack`, `\sp` и `\sb`, так что вам не нужно вводить `‘`, `’`, `[`, `]`, `^` и `_`, соответственно.

Тем, кто реализует TEX на вычислительных системах, которые не имеют 95 символов, представляемых внешне, советуем придерживаться следующих рекомендаций. (a) Максимально придерживаться условных обозначений ASCII. (b) Удостовериться в том, что коды '041 – '046, '060 – '071, '141 – '146 и '160 – '171 присутствуют и что каждый непредставляемый внутренний код ведет к представляемому коду, когда прибавляется или вычитается *´100*, тогда можно вводить и выводить все 256 кодов. (c) Сотрудничать с теми, кто испытывает подобные ограничения, чтобы всем придерживаться одинаковой политики. (См. приложение J — информация о группе пользователей TEX)

Только очень немногие условные соглашения о символьных кодах постоянно встроены в TEX. Почти все можно изменить с помощью форматного пакета, который устанавливает таблицы `\catcode`, `\mathcode`, `\uccode`, `\lccode`, `\sfcode` и `\delcode` и изменяет параметры типа `\escapechar`. Таким образом, с рукописью, подготовленной на TEX'е в Дании , можно работать, например, в Калифорнии, и наоборот, хотя в различных странах могут использоваться совершенно различные условные обозначения. Единственные символьные коды, которые TEX действительно “знает”, таковы: (1) `INITEX` инициализирует кодовые таблицы, как описано в приложении B; во всех реализациях TEX'а делается одинаковая инициализация. (2) TEX использует символьные коды `␣+-.,‘’"<=>0123456789ABCDEF` в своих синтаксических правилах (главы 20, 24 и приложение H), а также использует большинство строчных и прописных букв в своих ключевых словах `pt`, `to`, `plus` и т. д. Во всех реализациях TEX'а используются одинаковые коды и ключевые слова. Например, когда TEX реализуется для клавиатур с кириллицей , букве `П` должен быть присвоен код *´160*, а `Т` — код *´164* , так что “`ПТ`” означает “`pt`”, либо должны быть определены дополнительные команды, чтобы то, что видит TEX, было эквивалентно ключевым словам, которые ему необходимы. (3) Операции `\number`, `\romannumeral`, `\the` и `\meaning` могут генерировать буквы, цифры, пробелы, десятичные точки, знаки минус, двойные кавычки, двоеточие и знаки `>`. Во всех реализациях TEX'а генерированные коды одинаковы. (4) Команды `\hyphenation` и `\pattern`, описанные в приложении H, дают специальную интерпретацию десяти цифрам и символам “`.`” и “`-`”. (5) Коды для четырех символов `$ . { }` вставля-

ются тогда, когда TEX исправляет некоторые ошибки, скобки вставляются перед и после программы `\output`. Этим элементам даются специальные номера категорий, поэтому неважно, имеют ли эти символы значения в начальном TEX'е или нет. (6) Имеются специальные соглашения о представлении символов 0 – 255 в шестнадцатиричной форме `^^00` – `^^FF`, объясненные в главе 8. Это соглашение пиемлемо и в случае, когда `^` — любой символ кода 7. По этому соглашению вывод текста производится только когда представляются символы кода больше или равного 128, которые установщик TEX'а решил не выводить непосредственно.

Наборы кодов, полученные изменением стандартных наборов, как показано выше или другими заменами, являются нестандартными.

Code sets obtained by modifying the standard as shown above or by other replacements are nonstandard.

— ASA SUBCOMMITTEE X3.2, *American Standard Code for Information Interchange (1963)*

То, как в Стэнфорде и в ДЕК'е используют контрольные символы ASCII, является нарушением Стандартной кодировки США, но, надо полагать, никакой Генеральный Прокурор не прибежит и не арестует людей, которые нажимают CTR-T на своих компьютерах.

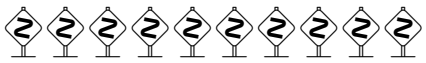
Both the Stanford and DEC uses of the ASCII control characters are in violation of the USA Standard Code, but no Federal Marshal is likely to come running out and arrest people who type control-T to their computers.

— BRIAN REID, *SCRIBE Introductory User's Manual* (1978)

D

Опасные трюки

TEX был разработан для того, чтобы выполнять обычные задания по набору текста: делать абзацы и страницы. Но лежащие в его основе механизмы, которые облегчают обычный набор текста — например, боксы, клей, штрафы и макрокоманды — обладают исключительной гибкостью. И вот люди открыли тайные способы заставить TEX делать различные хитрости, которые автор первоначально и не имел в виду. Обычно такие "умные" конструкции не считаются примерами "высокого TEX'а", но многие из них оказались полезными и наглядными и заслуживают того, чтобы о них знали (по крайней мере, некоторые знатоки). Цель данного приложения — привести умелых и/или смелых читателей на землю обетованную TEXанию.



Просим не читать этот материал до тех пор, пока полностью не освоите начальный TEX. После того, как вы прочтете и поймете все секреты, описанные ниже, вы будете знать все комбинации изощренных команд TEX'а и нередко испытаете искушение писать загадочные макрокоманды. Помните, однако, что всегда существует более хороший и более простой способ сделать что-то, чем тот, который первым пришел вам в голову. Может быть, вам и не нужно будет обращаться ни к каким увертkам, т.к. TEX многое может делать сам. Сначала пробуйте применить простые решения.

1. "Безумные" макрокоманды. Если вы хотите написать сложные макрокоманды, то следует изучить многие тонкости из главы 20. Команды TEX'а делятся на две основные категории: "раскрываемые" и "нераскрываемые". В первую категорию входят все макрокоманды и проверки `\if...\fi`, а также такие специальные операции, как `\the` и `\input`, в то время как во вторую категорию входят примитивные команды, перечисленные в главе 24. Раскрытие раскрываемых элементов происходит в "пасти" TEX'а, а примитивные команды (включая присваивания) выполняются в его "желудке". Важное следствие этой структуры состоит в том, что невозможно переопределить команду или обновить регистр в то время, когда TEX раскрывает список элементов, например, команду `\message` или `\write`. Операции присваивания выполняются только тогда, когда TEX создает вертикальный, горизонтальный или математический список.

Например, можно поместить в абзац `\n` звездочек, просто написав команды `{\loop\ifnum\n>0 *\advance\n-1 \repeat}`. Но намного труднее определить `\asts`, состоящую в точности из `\n` последовательных звездочек. Если бы было известно, что `\n`, скажем, не больше 5, то можно было бы написать

```
\edef\asts{\ifcase\n\or*\or**\or***\or****\or*****\else\bad\fi}
```

так как TEX обрабатывает `\ifcase` в своей "пасти". Но для обобщенных `\n` нельзя использовать конструкцию `\edef\asts{\loop\ifnum\n>0 *\advance\n-1 \repeat}`, так как `\n` не изменяется во время `\edef`. Здесь требуется более сложная программа, например,

```
{\xdef\asts{}
  \loop\ifnum\n>0 \xdef\asts{\asts*}\advance\n-1 \repeat}
```

Приведем другое решение (оно быстрее, потому что регистры списков элементов

могут быть с помощью \the раскрыты быстрее, чем макрокоманды):

```
\newcount\m \newtoks\t \m=\n \t={}
\loop \ifnum\m>0 \t=\expandafter{\the\t *} \advance\m-1 \repeat
\edef\asts{\the\t}
```

Однако в этих решениях время выполнения пропорционально квадрату \n. Есть более быстрый способ выполнить это задание:

```
\begingroup\aftergroup\edef\aftergroup\asts\aftergroup{
\loop \ifnum\n>0 \aftergroup*\advance\n-1 \repeat
\aftergroup}\endgroup
```

Понятно? Команды \aftergroup указывают на то, что после группы сохраняется целый список других элементов! Этот способ имеет только один недостаток — он занимает \n ячеек во входном стэке и еще \n в стэке спасения TEX'а, так что может понадобиться специальная версия TEX'а, когда \n больше 150 или около того.

(Кстати, есть другой способ вставить \n звездочек в абзац: написать \setbox0=\hbox{*}\cleaders\copy0\hskip\n\wd0. Может показаться, что это самое быстрое решение из всех, но если рассмотреть его подробно, то оказывается, что оно не такое уж быстрое, поскольку генерирует в выходной dvi-файл по четыре байта на каждую звездочку, в то время как в других методах генерируется всего по одному байту на звездочку. Время ввода/вывода превышает время вычислений как в самом TEX'е, так и на следующих стадиях процесса печати.)

Задача, которую мы только что решили, может показаться специальным приложением: в конце концов, кому нужна команда, которая содержит переменное число звездочек? Но эти же принципы применимы и в других подобных случаях, когда надо создать, например, спецификацию переменной длины \parshape. Аналогично, многие из "игровых задач", которые решаются ниже, предназначены для того, чтобы проиллюстрировать приемы, которые могут быть использованы в реальных ситуациях.

Точные правила раскрывания приведены в главе 20. Лучше всего знакомиться с раскрывающим механизмом TEX'а, наблюдая его в действии и просматривая протокольный файл при \tracingmacros=2 и \tracingcommands=2. Один из интересных способов изменить порядок раскрывания — это использовать команду \expandafter. Конструкция

```
\expandafter\a\b
```

приводит к тому, что сначала раскрывается \b, а затем \a. А поскольку команда \expandafter является раскрываемой, конструкция

```
\expandafter\expandafter\expandafter\a\expandafter\b\c
```

приводит к тому, что сначала раскрывается \c, потом \b, а потом \a. (Следующий шаг

```
\expandafter\expandafter\expandafter\expandafter
 \expandafter\expandafter\expandafter\a
 \expandafter\expandafter\expandafter\b\expandafter\c\d
```

пожалуй, слишком длинный, чтобы пользоваться им).

Можно также эффективно использовать `\expandafter\a\b` даже тогда, когда `\a` не раскрываемое. Например, присваивание `\t=\expandafter{\the\t *}` в приведенном выше примере, раскрывая что-то после левой скобки, могло вторгнуться на территорию, где раскрывание обычно запрещено. Аналогично,

```
\t=\expandafter{\expandafter*\the\t}
```

тоже работает, а

```
\uppercase\expandafter{\romannumeral\n}
```

дает значение регистра `\n`, набранное большими римскими цифрами.

Приведем более интересный пример. Вспомним, что `\fontdimen1` — это "наклон на пункт" шрифта. Поэтому, например, `\the\fontdimen1\tenit` раскрывается в `0.25pt`, где категория символов "`pt`" равна 12. После макроопределений

```
{\catcode`p=12 \catcode`t=12 \gdef\\#1pt{#1}}
\let\getfactor=\\
\def\kslant#1{\kern\expandafter\getfactor\the\fontdimen1#1\ht0}
```

можно написать, например, `\kslant\tenit`, что раскрывается в `\kern0.25\ht0`. Если граница `\box0` наклонена на 0.25 горизонтальных единиц на каждую вертикальную единицу, то этот керн измеряется горизонтальным расстоянием, на которое перекашивается верхний край бокса относительно края, находящегося на базовой линии. Вычисление `\kslant` полностью выполняется в "пасти" TEX'а. Таким образом, "пасть" умеет делать некоторые сложные вещи, хотя и не может присваивать новые значения. (Кстати, здесь применялся косвенный способ, чтобы определить команду `\getfactor`, когда символ `t` имел категорию 12, так как имена команд обычно состоят только из букв. Альтернативная конструкция

```
{\catcode`p=12 \catcode`t=12
  \csname expandafter\endcsname\gdef
  \csname getfactor\endcsname#1pt{#1}}
```

тоже работала бы, так как `\csname` и `\endcsname` не содержат ни `p`, ни `t`!)

Механизм, с помощью которого TEX определяет аргументы макрокоманды, может применяться неожиданными способами. Предположим, например, что `\t` — это регистр списка элементов, который содержит некоторый текст. Мы хотим определить, имеется ли в этом тексте хотя бы одна звездочка ($*_{12}$). Вот один из способов сделать это:

```
\newif\ifresult % для результата вычисленной проверки
\def\atest#1{\expandafter\a\the#1*\atest\a}
\long\def\a#1*#2#3\a{\ifx\atest#2\resultfalse\else\resulttrue\fi}
```

Теперь после `\atest\t` значение команды `\ifresult` будет либо `\iftrue`, либо `\iffalse`, в зависимости от того, содержит `\t` звездочку или нет. (Вам понятно, почему?) Это же можно сделать и более изящно, используя `\futurelet`, чтобы заглянуть вперед:

```
\def\btest#1{\expandafter\b\the#1*\bb}
\long\def\b#1*{\futurelet\next\bb}
\long\def\bb#1\bb{\ifx\bb\next\resultfalse\else\resulttrue\fi}
```

В обоих случаях решение работает, если `\t` содержит элементы команд вместе с символьными элементами , но не содержит специальных команд `\atest`, `\a` и `\bb`. Отметим, однако, что звездочка "спрятана", если она находится внутри группы `{...}` . Проверка ограничивается звездочками на нулевом уровне вложенности. Регистр списка элементов всегда сбалансирован относительно группирования, поэтому нет опасности, что проверка приведет к сообщению об ошибке в связи с пропущенными или лишними скобками.

Идеи из предыдущего абзаца можно использовать для того, чтобы решить проблему обобщенного математического форматирования. Надо сделать так, чтобы конструкции `$$`α`$$` , `$$`α`\eqno`β`$$` и `$$`α`\leqno`β`$$` приводили к вызову макрокоманды `$$\generaldisplay$$` и чтобы при этом `\eq` было определено как α. Кроме того, проверка `\ifeqno` должна быть истиной, когда присутствует уравнение номер β , а `\ifleqno` должна быть истиной в случае `\leqno`. Когда β присутствует, он должен храниться в `\eqn`. Здесь α и β являются произвольно сбалансированными списками элементов, которые на нулевом уровне вложенности содержат либо `\eqno`, либо `\leqno`. Необходимые маневры выполняются следующими макрокомандами:

```
\newif\ifeqno \newif\ifleqno \everydisplay{\displaysetup}
\def\displaysetup#1$${\displaytest#1\eqno\eqno\displaytest}
\def\displaytest#1\eqno#2\eqno#3\displaytest{%
  \if!#3!\ldisplaytest#1\leqno\leqno\ldisplaytest
  \else\eqnotrue\leqnofalse\def\eqn{#2}\def\eq{#1}\fi
  \generaldisplay$$}
\def\ldisplaytest#1\leqno#2\leqno#3\ldisplaytest{\def\eq{#1}%
  \if!#3!\eqnofalse\else\eqnotrue\leqnotrue\def\eqn{#2}\fi}
```

Анализ всех трех случаев `$$`α`$$`, `$$`α`\eqno`β`$$` и `$$`α`\leqno`β`$$` показывает, что гарантированы правильные действия. Параметр `#3` в проверках `\if!#3!` будет либо пустым, либо `\eqno`, либо `\leqno`. Таким образом, условие будет ложью (и второй знак `!` будет пропущен), если только `#3` не будет пустым.

Возвращаясь к проблеме `*` в `\t`, предположим, что необходимо рассматривать `*` на всех уровнях вложенности. Тогда нужно использовать более медленную программу:

```
\def\ctest#1{\resultfalse\expandafter\c\the#1\ctest}
\def\c{\afterassignment\cc\let\next= }
\def\cc{\ifx\next\ctest \let\next\relax
  \else\ifx\next*\resulttrue\fi\let\next\c\fi \next}
```

Здесь `\afterassignment` было использовано для того, чтобы сохранить контроль после обычного `\let` (не `\futurelet`), знак `=` гарантирует, что за одну команду `\c` будет "заглатываться" только один элемент. Эту программу можно было бы очевидным образом модифицировать, чтобы считать общее количество звездочек и/или элементов в `\t`. Обратите внимание на `\let\next` в `\cc`. Должно быть понятно, почему альтернативный вариант

```
\def\cc{\ifx\next\ctest\else\ifx\next*\resulttrue\fi\c\fi}
```

не будет срабатывать. (Последнее `\c` будет всегда поглощать `\fi`.)

Элементы пробелов иногда бывают необычными, поэтому они требуют особого внимания. Следующая макрокоманда `\futurenonspacelet` ведет себя в сущности так же, как `\futurelet`, за тем исключением, что отбрасывает любые явные или неявные пробелы, которые встречаются перед непробелами:

```
\def\futurenonspacelet#1{\def\cs{#1}%
  \afterassignment\stepone\let\nexttoken= }
\def\\{\let\stoken= } \\ % теперь \stoken является элементом пробела
\def\stepone{\expandafter\futurelet\cs\steptwo}
\def\steptwo{\expandafter\ifx\cs\stoken\let\next=\stepthree
  \else\let\next=\nexttoken\fi \next}
\def\stepthree{\afterassignment\stepone\let\next= }
```

Операция типа `\futurenonspacelet` полезна, например, при реализации макрокоманд, имеющих переменное число аргументов.

Отметим, что `\def\stepthree#1{\stepone}` здесь не срабатывает из-за правила, что элемент $␣_{10}$ пропускается, если иначе он рассматривался бы как неограниченный аргумент. Из-за этого правила бывает трудно отличить явные пробелы от неявных. Ситуация особенно сложная, потому что можно использовать `\uppercase`, чтобы создавать "забавные пробелы" типа $*_{10}$. Например, команды

```
\uccode`~=`* \uppercase{\uppercase{\def\fspace{ }\let\ftoken= } }
```

делают `\fspace` макрокомандой, которая раскрывается в забавный пробел, и делают `\ftoken` неявным забавным пробелом. (Проверки `\if\fspace*`, `\if\ftoken*`, `\ifcat\fspace\stoken` и `\ifcat\ftoken\stoken` будут истинными, если предполагается, что `*` имеет категорию 12. Но если `*` имеет категорию 10, то `\if\fspace*` будет ложью, потому что TEX все вновь созданные элементы пробелов приводит к норме $␣_{10}$, как объяснялось в главе 8.) Поскольку поведение различных видов пробелов почти идентично, мы не будем здесь вдаваться в детали. †

---

† Следующая небольшая программа предназначена для пользователей, которые настаивают на изучении всех подробностей. Макрокоманда `\stest` определяет, является ли первый элемент данного регистра списка элементов пробелом, как это определено в главе 24. Если да, то макрокоманда решает, является ли элемент "забавным" или нет, то есть отличается ли его символьный код от кода ⟨пробел⟩ ASCII. Макрокоманда также определяет, является элемент явным или неявным.

```
\newif\ifspace \newif\iffunny \newif\ifexplicit
\def\stest#1{\expandafter\s\the#1 \stest}
\def\s{\futurelet\next\ss}
\def\ss{\ifcat\noexpand\next\stoken\spacetrue
   \ifx\next\stoken\let\next=\sss\else\let\next=\ssss\fi
  \else\let\next=\sssss\fi \next}
\long\def\sss#1 #2\stest{\funnyfalse
  \def\next{#1}\ifx\next\empty\explicittrue\else\explicitfalse\fi}
\long\def\ssss#1#2\stest{\funnytrue
  \ifcat\noexpand#1\noexpand~\explicitfalse % active funny space
  \else{\escapechar=\if*#1`?\else`*\fi
    \if#1\string#1\global\explicittrue\else\global\explicitfalse\fi}\fi}
\long\def\sssss#1\stest{\spacefalse}
```

Аргумент команды `\write` раскрывается, когда встречается `\shipout`, но иногда это нежелательно. Приведем макрокоманду (предложенную Тоддом Алленом) , которая подавляет все раскрытия, вставляя `\noexpand` перед каждой командой или активным символом. Макрокоманда предполагает, что `~` является активным символом и что записываемые элементы не содержат неявных пробелов или скобок. Забавные пробелы заменяются на обычные.

```
\long\def\unexpandedwrite#1#2{\def\finwrite{\write#1}%
  {\aftergroup\finwrite\aftergroup{\sanitize#2\endsanity}}}
\def\sanitize{\futurelet\next\sanswitch}
\def\sanswitch{\ifx\next\endsanity
  \else\ifcat\noexpand\next\stoken\aftergroup\space\let\next=\eat
   \else\ifcat\noexpand\next\bgroup\aftergroup{\let\next=\eat
    \else\ifcat\noexpand\next\egroup\aftergroup}\let\next=\eat
     \else\let\next=\copytoken\fi\fi\fi\fi \next}
\def\eat{\afterassignment\sanitize \let\next= }
\long\def\copytoken#1{\ifcat\noexpand#1\relax\aftergroup\noexpand
  \else\ifcat\noexpand#1\noexpand~\aftergroup\noexpand\fi\fi
  \aftergroup#1\sanitize}
\def\endsanity\endsanity{}
```

Как и прежде, активное использование команды `\aftergroup` в определении макрокоманды `\unexpandedwrite` означает, что параметр `#2` не должен включать более чем примерно 150 элементов.

2. Списочные макрокоманды. Следующие несколько макрокоманд, которые мы рассмотрим, могут быть использованы для того, чтобы хранить списки информации в виде

$$\backslash\backslash\{\langle\text{элемент}_1\rangle\}\backslash\backslash\{\langle\text{элемент}_2\rangle\}\ \ldots\ \backslash\backslash\{\langle\text{элемент}_n\rangle\}$$

где каждый ⟨элемент⟩ является сбалансированным списком элементов. Команду без параметров, текст замены которой имеет такой вид, можно назвать *списочной макрокомандой*. Пустая списочная макрокоманда имеет $n = 0$ и называется `\empty`.

Можно легко добавлять новые элементы к любому концу списочной макрокоманды, а также сцеплять списочные макрокоманды друг с другом, например, таким образом:

```
\toksdef\ta=0 \toksdef\tb=2 % token list registers for temp use
\long\def\leftappenditem#1\to#2{\ta={\\{#1}}\tb=\expandafter{#2}%
  \edef#2{\the\ta\the\tb}}
\long\def\rightappenditem#1\to#2{\ta={\\{#1}}\tb=\expandafter{#2}%
  \edef#2{\the\tb\the\ta}}
\def\concatenate#1=#2&#3{\ta=\expandafter{#2}\tb=\expandafter{#3}%
  \edef#1{\the\ta\the\tb}}
```

И наоборот, левый элемент списка может быть удален или помещен в команду макрокомандой `\lop`, которая определяется весьма любопытным образом:

```
\def\lop#1\to#2{\expandafter\lopoff#1\lopoff#1#2}
\long\def\lopoff\\#1#2\lopoff#3#4{\def#4{#1}\def#3{#2}}
```

Например, если `\l` раскрывается в список `\\{a\b}\\{c}\\{{d}}`, то макрокоманда `\lop\l\to\z` заставляет `\l` раскрываться в `\\{c}\\{{d}}`, а `\z` — в `a\b`. Операцию `\lop` следует использовать только тогда, когда `\l` будет непустым, иначе возникнет ошибка. Чтобы проверить, является ли `\l` пустым, нужно просто написать `\ifx\l\empty`.

Детали программирования макрокоманды `\lop` показывают, почему отдельные элементы были заключены в группы. Для многих целей достаточно списка более простого типа, в котором группирование опускается, а в конце появляются дополнительные `\\`. Например, можно определить

```
\long\def\lopoff\\#1\\#2\lopoff#3#4{\def#4{#1}\def#3{\\#2}}
```

и результат будет почти такой же, как прежде. В этом случае пустая списочная макрокоманда раскрывается в `\\`. Однако, новая `\lop`, получающаяся из этой новой макрокоманды `\lopoff`, также удаляет пару скобок, если крайний левый элемент оказывается группой. Чтобы избежать таких аномалий, в общую схему включаются дополнительные скобки.

Рассмотренные нами примеры все еще не объяснили, почему в общей схеме появляются `\\`. Может показаться, что самого группирования было бы достаточно. Но в действительности разделители `\\` чрезвычайно полезны, потому что можно определить `\\` как любую желаемую макрокоманду с одним аргументом, и после этого *выполнить* список! Например, приведем способ подсчета количества элементов:

```
\def\cardinality#1\to#2{#2=0 \long\def\\##1{\advance#2 by1 }#1}
```

(Предполагается, что параметр `#2` — имя count-регистра.) А еще можно взять списочную макрокоманду и центрировать все элементы на отдельных строках внутри `\vbox`:

```
\def\centerlist#1{\def\\##1{\relax##1\cr}%
  \vbox{\halign{\hfil##\hfil\cr#1}}}
```

Конкретный элемент может быть выбран по номеру позиции слева:

```
\def\select#1\of#2\to#3{\def#3{\outofrange}%
  \long\def\\##1{\advance#1-1 \ifnum#1=0 \def#3{##1}\fi}#2}
```

(Здесь `#1` — count-регистр, `#2` — списочная макрокоманда, а `#3` — команда.) И так далее — можно придумать сотни других применений.†

TEX эффективно выполняет все предшествующие операции в том смысле, что время выполнения пропорционально длине включенной списочной макрокоманды. Естественно задать вопрос, можно ли с такой же эффективностью удалить правый крайний элемент, так как изолировать последний элемент списка не так-то просто. По-видимому, не существует способа удалить n-ый элемент из n-элементного списка за число шагов порядка n, сохраняя полную общность, если только не используется прием `\aftergroup` (с помощью которого мы создали макрокоманду, которая раскрывается в n звездочек), а прием `\aftergroup` не кажется привлекательным применительно к списку, поскольку список может быть

† Понятие списочной макрокоманды тесно связано с понятием списочных процедур в языках программирования; см. *Communications of the ACM* **7** (1964), 280.

довольно длинным.‡ Однако, если в списке только нераскрываемые элементы, то можно довольно успешно удалить правый крайний элемент :

```
\def\deleterightmost#1{\edef#1{\expandafter\xyzzy#1\xyzzy}
\long\def\xyzzy\\#1#2{\ifx#2\xyzzy\yzzyx
  \else\noexpand\\{#1}\fi\xyzzy#2}
\long\def\yzzyx#1\xyzzy\xyzzy{\fi}
```

Тщательное изучение этого примера показывает, что "пасть" TEX'а может выполнять рекурсивные операции, если, конечно, макрокоманды заданы достаточно умело.

Содержимое `\count`-регистра можно легко преобразовать в десятичную форму и запомнить в какой-нибудь команде. Например, если `\n` является регистром, то операция `\edef\csn{\the\n}` помещает его значение в `\csn`. И наоборот, значение из `\csn` может быть помещено обратно в `\n`, если написать просто `\n=\csn`. Обычно это преобразование делается только для того, чтобы свести к минимуму использование `\count`-регистров, так как их в TEX'е 256, но десятичное представление, подобное результату раскрытия `\csn`, может запоминаться в списочной макрокоманде, что полезно в некоторых приложениях. Кстати, можно точно проверить, будет ли такой номер команды нулем:

```
\if0\csn⟨истинный текст⟩\else⟨ложный текст⟩\fi
```

Эта макрокоманда работает потому, что ⟨истинный текст⟩ будет игнорировать лишние цифры ненулевого номера.

Есть еще один прием, похожий на списочные макрокоманды, который можно использовать для хранения неупорядоченных множеств команд. Иногда удобно убирать скобки. Например,

```
\def\l{\\\alpha\\\beta\\\gamma}
```

определяет "множество макрокоманд" `\l`, которое представляет собой команды {`\alpha`, `\beta`, `\gamma`}. Простая конструкция проверяет, находится ли данная команда в множестве или нет:

```
\def\ismember#1\of#2{\resultfalse\def\given{#1}%
  \def\\##1{\def\next{##1}\ifx\next\given\resulttrue\fi}#2}
```

Другая эффективная, но не столь простая конструкция удаляет все команды, которые `\ifx`-эквивалентны данной команде:

```
\def\remequivalent#1\from#2{\let\given=#1%
  \ifx#2\empty\else\edef#2{\expandafter\plugh#2\plugh}\fi}
\def\plugh\\#1#2{\ifx#1\given\else\noexpand\\\noexpand#1\fi
  \ifx#2\plugh\hgulp\fi\plugh#2}
\def\hgulp\fi\plugh\plugh{\fi}
```

3. Дословная распечатка. Начальный TEX имеет макрокоманду `\dospecials`, которая в сущности является множественной макрокомандой, представляющей множество всех символов, которые имеют специальный номер категории. (Команда

‡ Заинтересованный читатель может получить удовольствие, создавая макрокоманду, которая удаляет k-ый элемент из n-элементной списочной макрокоманды. `\l` за $O(n \log n)$ шагов, если дано k и `\l`, не используя `\aftergroup`.

`\do` играет роль `\\` в предыдущем объяснении.) Поэтому легко изменить категорию всех специальных символов на 12 (другие):

```
\def\uncatcodespecials{\def\do##1{\catcode`##1=12 }\dospecials}
```

Это работает, даже если множество специальных символов было изменено, при условии, что `\dospecials` модифицировано для текущего множества.

Только что определенная операция `\uncatcodespecials` имеет большое значение, когда автоматические средства TEX'а должны быть временно блокированы. Предположим, что мы хотим создать распечатку некоторого машинного файла, воспроизводящего символы и пробелы в точности так же, как они появляются в этом файле. Чтобы сделать задачу более интересной, давайте также печатать номера строк напротив каждой строки, как это сделано в распечатке `story.tex` на странице 24. Чтобы упростить задачу, давайте предположим, что файл содержит только стандартные печатные символы ASCII: знаки табуляции, протяжки страницы и тому подобное отсутствуют. Надо придумать такую макрокоманду `\listing`, чтобы, например, `\listing{story}` вставляло распечатку файла `story.tex` в рукопись, после чего нормальные соглашения TEX'а восстанавливались. Распечатка должна быть сделана шрифтом `\tt`. Этим требованиям отвечает следующая макрокоманда:

```
\def\listing#1{\par\begingroup\setupverbatim\input#1 \endgroup}
```

Обратите внимание, как команда `\endgroup` будет здесь тонко "отключать" все причудливые вещи, которые включает `\setupverbatim`. Обратите внимание также на то, что команды `\input#1 \endgroup` не будут перечислены в списке дословно, даже если они следуют за `\setupverbatim`, поскольку они введены считывающим механизмом TEX'а, когда раскрывалась макрокоманда `\listing` (то есть до того, как была задана дословная передача.)

А что будет делать команда `\setupverbatim`? Ну, она должна включить `\obeylines`, так как при этом автоматически вставляется `\par` в конце каждой вводимой строки. Она должна включить `\uncatcodespecials`, чтобы специальные символы печатались сами по себе, а также включить `\obeyspaces`, чтобы учитывался каждый пробел. Надо внимательно рассмотреть все эти штуки, чтобы правильно понять, что они делают. (1) Макрокоманда начального TEX'а `\obeylines` заменяет номер категории у `^^M` на `\active`, а затем говорит `\let^^M=\par`. Так как `^^M` помещается в конце каждой строки, каждая строка благополучно заканчивается `\par`. Однако, `\obeylines` не говорит `\def^^M{\par}`, поэтому мы должны сделать все нужные изменения `\par` перед тем, как вызвать `\obeylines`. (2) Операция `\uncatcodespecials` заменяет пробел на символ категории 12, но шрифт `\tt` имеет символ ␣ в позиции ⟨пробел⟩, так что нам на самом деле $␣_{12}$ не нужен. (3) Макрокоманда `\obeyspaces` в приложении B просто заменяет символ ⟨пробел⟩ на категорию 13. Было определено, что активный символ $␣_{13}$ — это то же самое, что `\space`, макрокоманда, которая раскрывается в $␣_{10}$. Обычно именно это и нужно, так как пробелы в конструкциях типа `\hbox to 10 pt {...}` не создают никаких проблем. Но в нашей задаче это имеет нежелательный эффект, так как создает промежутки, на которые действует коэффициент пробела. Чтобы отменить это, нужно либо написать `\frenchspacing`, либо сделать новое определение, чтобы $␣_{13}$ был идентичным $\␣$. Второй вариант лучше, поскольку в первом пробелы в начале каждой строки будут отбрасываться.

Макрокоманда `\setupverbatim` должна обеспечивать размещение номера строки в позиции абзацного отступа. Мы можем побеспокоиться об этом, вводя переменную счетчика и используя `\everypar` следующим образом:

```
\newcount\lineno % количество перечисленных строк файла
\def\setupverbatim{\tt \lineno=0
  \obeylines \uncatcodespecials \obeyspaces
  \everypar{\advance\lineno by1 \llap{\sevenrm\the\lineno\ \ }}}
{\obeyspaces\global\let =\ } % активный пробел = командный пробел
```

Теоретически кажется, что это должно работать, но на практике происходит два сбоя. Один сбой, наиболее очевидный (по крайней мере, он становится очевидным, когда тестируется макрокоманда), состоит в том, что все пустые строки файла опускаются. Это объясняется тем, что команда `\par` в конце пустой строки не начинает новый абзац, потому что встречается вертикальная мода. Второй сбой не столь очевиден, так как происходит гораздо реже: шрифты `\tt` содержат лигатуры для испанской пунктуации, поэтому последовательности `?‘` и `!‘` будут печататься, соответственно, как ¿ и ¡. Оба эти недостатка можно исправить, вставив

```
\def\par{\leavevmode\endgraf} \catcode`\`=\active
```

перед `\obeylines` в макрокоманде `\setupverbatim` и определив ‘$_{13}$ следующим образом:

```
{\catcode`\`=\active \gdef`{\relax\lq}}
```

Аналогичную схему можно использовать, чтобы произвести дословные распечатки другими шрифтами, но нужно сделать активными большее количество символов, чтобы разрушить лигатуры и компенсировать символы, отсутствующие в ASCII.

В дополнение к дословной распечатке файла можно определить такую макрокоманду `\verbatim`, чтобы из `\verbatim{$это$ есть{\it!}}` получалось “`$это$ есть {\it!}`”. Менять номера категорий довольно опасно, потому что TEX штампует категорию на каждом символе, впервые считывая его из файла. Поэтому если бы `\verbatim` было определено конструкцией

```
\long\def\verbatim#1{⟨что-то⟩},
```

то аргумент `#1` был бы уже преобразован в список элементов к моменту, когда начинается ⟨что-то⟩. Изменения `\catcode` не повлияли бы на аргумент. Альтернативный вариант состоит в том, чтобы изменить категории перед просматриванием аргумента команды `\verbatim`:

```
\def\verbatim{\begingroup\tt\uncatcodespecials
  \obeyspaces\doverbatim}
\newcount\balance
{\catcode`<=1 \catcode`>=2 \catcode`\{=12 \catcode`\}=12
  \gdef\doverbatim{<\balance=1\verbatimloop>
  \gdef\verbatimloop#1<\def\next<#1\verbatimloop>%
    \if#1{\advance\balance by1
    \else\if#1}\advance\balance by-1
     \ifnum\balance=0\let\next=\endgroup\fi\fi\fi\next>>
```

Это работает, но медленно, и позволяет задавать дословную передачу только для текста, который имеет сбалансированные скобки. Для печати примеров в книге типа *The TEXbook* это не годится. (Приложение E содержит дословные макрокоманды, которые использовались в действительности.) Отметим также, что если эта макрокоманда `\verbatim{...}` появляется в аргументе другой макрокоманды типа `\centerline`, она не срабатывает, потому что номера категорий уже не могут быть изменены. Макрокоманда `\footnote` в приложении B предусмотрительно избегает преждевременного просмотра аргументов. Она довольно искусно использует `\bgroup` и `\egroup`, так что изменения номеров категорий разрешаются внутри сносок начального TEX'а.

С другой стороны, есть довольно быстрый способ преобразовать список элементов в почти дословную транскрипцию:

```
\long\def\verbatim#1{\def\next{#1}%
  {\tt\frenchspacing\expandafter\strip\meaning\next}}
\def\strip#1>{}
```

Элементы в этой конструкции "обнажаются", поскольку, например, `\meaning\next` может быть "`macro:->$это$ есть {\it !}`". Отметим, что пробел будет вставляться после командного слова `\it`, но в аргументе команды `\verbatim` никакого пробела не будет. Такая информация безвозвратно потеряна.

Одна из проблем, связанных с режимом дословной передачи, состоит в том, что его трудно остановить: если мы отключим все нормальные управляющие возможности TEX'а, то кончим тем, что "загоним себя в угол" и достигнем точки, откуда нет возврата. Макрокоманда `\listing` могла бы решить эту проблему, потому что конец файла возвращает к жизни старый список элементов. Другое возможное решение состоит в том, чтобы указать определенное количество строк, после которого режим дословной передачи должен закончиться. Иначе необходимо наложить на текст некоторые ограничения, то есть сделать определенные тексты не подлежащими печати в дословном режиме. Например, приведем метод, который печатает все между `\beginverbatim` и `\endverbatim`, предполагая только, что не нужно печатать команду `\endverbatim`:

```
\def\beginverbatim{\par\begingroup\setupverbatim\doverbatim}
{\catcode`\|=0 \catcode`\\=12 % | временный символ перехода
  |obeylines|gdef|doverbatim^^M#1\endverbatim{#1|endgroup}}
```

Эта конструкция предполагает, что `\beginverbatim` появляется в конце строки рукописного файла. Аргумент `#1` будет полностью считан в память TEX'а до того, как что-либо произойдет, так что было бы лучше, если бы общее количество дословного материала не было слишком объемным. Кстати, нет необходимости указывать, что эта макрокоманда является `\long`, потому что `\par`, вставляемые командами `\obeylines`, в действительности представляют собой `^^M`.

Другой способ состоит в том, чтобы сохранить один символ нетронутым. Можно делать определения так, что

`\verbatim`⟨символ⟩⟨текст⟩⟨символ⟩

будет дословно набирать ⟨текст⟩, где предполагается, что ⟨текст⟩ не содержит

повторяющегося ограничителя ⟨символ⟩:

```
\def\verbatim{\begingroup\setupverbatim\doverbatim}
\def\doverbatim#1{\def\next##1#1{##1\endgroup}\next}
```

4. Выборочная загрузка макрокоманд. Когда у вычислительной системы накапливается большая библиотека макрокоманд, начинаюь возникать интересные проблемы. Предположим, например, что файл `macs.tex` содержит строки

```
\let\italcorr=\/
\def\/{\unskip\italcorr}
```

потому что кто-то подумал, что было бы приятно разрешить возможный пробел перед примитивной командой `\/`. Это неплохо, за исключением того случая, когда `macs.tex` вводится дважды. Например, два других файла указывают `\input macs`. Когда эти строки обрабатываются во второй раз, `\italcorr` будет `\let`-равен макрокоманде, которая раскрывается в `\unskip\italcorr`, и можно догадаться, что произойдет: TEX войдет в бесконечный цикл, который можно остановить, только прервав программу вручную.

К счастью, есть простой способ справиться с этой проблемой, поместив подходящую блокировку в начале каждого файла, который может вводить такие аномалии:

```
\ifx\macsisloaded\relax\endinput\else\let\macsisloaded=\relax\fi
```

Тогда `\macsisloaded` будет неопределенным во время первого `\ifx`, но файл не будет считываться дважды. Для каждого файла следует, разумеется, использовать свою команду.

Другая сложность в работе с большими наборами макрокоманд состоит в том, что они занимают много места. Как хорошо было бы загрузить все макрокоманды, когда-либо придуманные всеми пользователями TEX'а! Но для этого может не хватить места, потому что емкость памяти TEX'а имеет предел. Вам придется сдержаться и загрузить только те макрокоманды, которые действительно необходимы.

Какой объем памяти занимает макрокоманда? Всего имеется четыре вида памяти: память элементов, память имен, память строк и память символов. (Если какая-либо из них переполняется, то нужно будет увеличить запрос на, соответственно, размер памяти макрокоманд, размер хэш-памяти, количество строк и/или размер накопителя; см. главу 27.) Память элементов самая важная. Макрокоманда занимает по одной ячейке памяти элементов для каждого элемента в своем определении, включая `{` и `}`. Например, сравнительно короткое определение

```
\def\example#1\two{\four}
```

занимает пять элементов: `#1`, `two`, $\{_1$, `four` и $\}_2$. Каждая команда также занимает одну ячейку памяти имен, одну ячейку памяти строк и столько ячеек памяти символов, сколько символов имеется в имени (в случае с `\example` - семь). Память символов сравнительно дешевая: четыре символа (а в некоторых случаях пять) будут занимать внутри машины такое же количество бит, как одна ячейка памяти элементов. Поэтому вы много не сэкономите, выбирая короткие имена макрокоманд.

TEX сообщит, насколько вы приблизились к переполнению объема текущей памяти, если вы напишете `\tracingstats=1`. Например, во время одного из прогонов, которые автор делал при проверке пробных оттисков этого приложения, была сообщена следующая статистика:

```
Here is how much of \TeX's memory you used:
 197 strings out of 1077
 1616 string characters out of 9424
 3434&13407 words of memory out of 25000&33001
 1244 multiletters control sequences out of 2100
```

Отсюда следует, что имеется достаточно места для дополнительных макрокоманд: $33001-13407 = 19594$ — неиспользованные ячейки памяти элементов, $2100-1244 = 856$ — памяти имен, $1077-197 = 880$ — памяти строк и $9424-1616 = 7808$ — памяти символов. Но был использован довольно большой TEX и загружены только макрокоманды приложений B и E. При других обстоятельствах могло понадобиться экономить место.

Один из простых способов избежать загрузки слишком большого количества макрокоманд — это хранить короткие файлы макрокоманд и `\input` только те файлы, которые нужны. Но пользоваться короткими файлами может оказаться неудобным. Иногда можно сделать по-другому. Предположим, например, что файл содержит пять необязательных классов макрокоманд, называемых `A`, `B`, `C`, `D` и `E`, и что типичному пользователю понадобится, вероятно, не больше двух-трех из этих пяти классов. Давайте создадим макрокоманду `\load` так, что, например, `\load{macs}{AC}` будет загружать файл `macs.tex`, включающий варианты `A` и `C`, но не `B`, `D` или `E`. Следующая макрокоманда `\load` преобразует второй аргумент в макрокоманду множества, называемую `\options`:

```
\def\load#1#2{\let\options=\empty \addoptions#2\end \input#1 }
\def\addoptions#1{\ifx#1\end \let\next=\relax
  \else\let\\=\relax\edef\options{\options\\#1}%
   \let\next=\addoptions \fi \next}
```

Внутри файла `macs.tex` та часть кодировки, которая должна загружаться только, скажем, в версии `B`, может быть заключена внутри `\ifoption B ... \fi`, где `\ifoption` определяется таким образом:

```
\def\ifoption#1{\def\\##1{\if##1#1\resulttrue\fi}%
  \resultfalse \options \ifresult}
```

(Это простое применение идей, изложенных в этом приложении ранее.)

Однако схема `\ifoption...\fi` не очень понятна, потому что она требует, чтобы все макрокоманды в необязательной части были правильно вложенными относительно `\if...` и `\fi`. Саму макрокоманду `\ifoption` невозможно легко определить в таком месте! Есть более подходящая схема, которая к тому же работает быстрее, основываясь на изменениях номеров категорий . Эта идея (предложенная Max Díaz) требует, чтобы крайний левый непустой символ на каждой строке был либо `\` либо `{`. Сделать это обычно не составляет труда. Кроме того, какой-либо другой символ, скажем `~`, резервируется. Тогда тексту, который должен быть загружен только в версии `B`, предшествует строка "`\beginoption B`", а следует за ним

строка, в которой говорится "`~endoptionalcode`". `\catcode` для `~` задается равным 14 (символ комментария), следовательно, строка `~endoptionalcode` не будет иметь никакого влияния, если код не пропускается. Макрокоманда `\beginoption` действует таким образом:

```
\def\beginoption#1{\ifoption#1\else\begingroup\swapcategories\fi}
\def\swapcategories{\catcode`\\=14 \catcode`\{=14 \catcode`\~=0 }
\let\endoptionalcode=\endgroup
\catcode`\~=14
```

Когда категории будут изменены, все строки с большой скоростью будут пропускаться до тех пор, пока не встретится команда `~endoptionalcode`, а затем все будет восстановлено в прежнее состояние. По этой схеме материал, который должен быть загружен только и в версии `B`, и в версии `D`, может начинаться с `\beginoption B` и `\beginoption D`, а материал, который должен быть загружен либо в версии `B`, либо в версии `D` (либо в обеих), может начинаться с

```
\beginoption B
~oroption D
```

если мы определим `\oroption#1` как сокращение для `\ifoption#1\endgroup\fi`.

Иногда подходит другой тип выборочной загрузки, основанный на том, определена или нет конкретная команда. По этой схеме, если команда не определена, она должна оставаться неопределенной и не занимать места в памяти TEX'а. Это можно сделать очень просто, а именно:

```
\ifx\cs\undefined ... \fi
```

(предполагается, что `\undefined` никогда не было определено). TEX не помещает неопределенные команды в свои внутренние таблицы, если они следуют за `\ifx` или если они встречаются во время пропуска условного текста. Можно использовать эту идею, например, при подготовке библиографии для статьи путем считывания скомпонованного должным образом библиографического файла. Будут загружаться только те элементы, которые соответствуют определенным командам.

5. Трюки со скобками. Некоторые операции TEX'а зависят от группирования, и если вы попытаетесь делать какие-то хитрые штуки, вам захочется точно знать, что значит это группирование. Например, команды plain TEX'а `\bgroup` и `\egroup` являются "неявными скобками" , потому что они определены так:

```
\let\bgroup={  \let\egroup=}
```

Это значит, что можно включать их в тексты замены определений, не беспокоясь о том, как они вложены. Например, макрокоманды

```
\beginbox{\setbox0=\hbox\bgroup}
\def\endbox{\egroup\copy0 }
```

позволяют создать бокс между `\beginbox` и `\endbox`. Они будут вести себя почти так же, как

```
\def\beginbox#1\endbox{\setbox0=\hbox{#1}\copy0 }
```

но с тремя важными отличиями. (1) Первый вариант допускает изменение номеров категорий внутри бокса. (2) Первый вариант быстрее, потому что не нужно просматривать содержимое бокса как в качестве аргумента, так и в качестве последовательности действительных команд. (3) Первый вариант занимает меньше памяти, потому что не нужно хранить аргументы. Поэтому предпочтение обычно отдается первому варианту.

Для простоты обсуждения предположим, что только `{` имеет категорию 1 и только `}` имеет категорию 2, хотя в действительности в качестве ограничителей групп могут использоваться любые символы. Вложенность групп является решающим моментом в двух основных видах деятельности TEX'а: (а) когда TEX просматривает ⟨сбалансированный текст⟩, например, когда он формирует текст замены макрокоманды, параметр или переменную списка элементов; (б) когда TEX должен определить, являются ли элементы `&`, `\span`, `\cr` или `\crcr` концом элемента внутри выравнивания.

В пасти TEX'а имеется два счетных механизма для работы с вложенностью. "Основной счетчик" увеличивается на 1 для каждого $\texttt{\{}_1$, который просматривается TEX'ом, и уменьшается на 1 для каждого $\texttt{\}}_2$. "Сбалансированный счетчик" аналогичен ему, но на него оказывают влияние только явные элементы $\texttt{\{}_1$ и $\texttt{\}}_2$, которые действительно заносятся в формируемый список элементов. Основной счетчик уменьшается на 1, когда TEX оценивает буквенную константу `‘{`, и увеличивается на 1, когда TEX оценивает `‘}`, так что, когда оценены две такие константы, общее изменение равно нулю. Как следствие этих правил, некоторые конструкции имеют следующий эффект:

	Основной счетчик		*Сбалансированный счетчик*	
Ввод	*раскрытый*	*нераскрытый*	*раскрытый*	*нераскрытый*
`{`	1	1	1	1
`\bgroup`	0	0	0	0
`\iffalse{\fi`	1	1	0	1
`\ifnum0=‘{\fi`	0	1	0	1

Два последние случая не производят элементов начала группы при раскрытии, но воздействуют на основной счетчик, как это показано. Таким образом, например,

```
\def\eegroup{\ifnum0=‘{\fi}}
```

делает поведение `\eegroup` весьма сходным с `\egroup`, но раскрытие `\eegroup` к тому же уменьшает основной счетчик.

При работе с выравниванием используется только основной счетчик, а сбалансированный не используется. Элемент выравнивания заканчивается первым `&` , `\span`, `\cr` или `\crcr`, который появляется, когда основной счетчик имеет значение, которое было в счетчике в начале ввода. Так, например, любопытная конструкция

```
\halign{\show\par#\relax\cr
  \global\let\par=\cr
  {\global\let\par=\cr}\cr
  \par}
```

заставляет TEX выполнять три команды `\show`, в которых соответствующими значениями указанного `\par` являются `\par`, `\relax` и `\cr`. Аналогично, каждый шаб-

лон в преамбуле выравнивания заканчивается первым `&`, `\cr` или `\crcr`, который появляется на уровне основного счетчика, действующего в начале элемента. Поэтому элементы `&` , `\cr` и `\crcr` могут появляться внутри шаблона выравнивания, если они скрываются за скобками (например, если они появляются в определении).

Эти факты позволяют нам сделать два довольно удивительных вывода. (1) Если элемент выравнивания имеет вид

$$\alpha\,\texttt{\textbackslash iffalse\{\textbackslash fi}\,\beta\,\texttt{\textbackslash iffalse\}\textbackslash fi}\,\gamma\,,$$

то β может включать элементы `&` и `\cr`, которые не являются локальными в группе.* (2) Конструкция

```
{\span\iffalse}\fi
```

появляющаяся в преамбуле, вносит `{` в шаблон, не меняя значение основного счетчика. Таким образом, она очень похожа на `\bgroup`, за тем исключением, что производит $\texttt{\{}_1$ явно. Если вы поняли (1) и (2), то согласитесь с тем, что данное приложение заслуживает своего названия.

6. Маневры с боксами. А теперь обратимся от синтаксиса к семантике, то есть от пасти TEX'а к его желудочно-кишечному тракту. Иногда какой-нибудь символ нужно напечатать жирным шрифтом, а он имеется только в нормальном виде. В таких случаях иногда можно выйти из положения с помощью "жирного шрифта для бедняков": нужно несколько раз напечатать этот символ с нормальной шириной почти на одном месте с очень небольшим сдвигом. Следующая макрокоманда набирает свой аргумент три раза в трех разных, но близких местах, равноудаленных друг от друга, но результат занимает столько же места, сколько его было бы занято, если бы `\pmb` было просто `\hbox`:

```
\def\pmb#1{\setbox0=\hbox{#1}%
  \kern-.025em\copy0\kern-\wd0
  \kern.05em\copy0\kern-\wd0
  \kern-.025em\raise.0433em\box0 }
```

Например, `\pmb{$\infty$}` дает ∞. Результаты будут немного **нечеткие**, и они, конечно, не соответствуют реальным символам, если они присутствуют, но "шрифт для бедняков" все же лучше, чем ничего и иногда в трудных ситуациях он может вас выручить.

Когда вы помещаете что-нибудь в регистр бокса, вам не обязательно надо помещать содержимое этого регистра в ваш документ. Так, можно написать макрокоманды, которые делают эксперименты "за кулисами", пробуя различные возможности перед тем, как принять определенное решение. Предположим, например, что вы набираете текст на двух языках, и вам хотелось бы выбрать такую ширину колонки, чтобы в обоих случаях было получено одинаковое количество

---

* Список элементов α не должен быть, однако, пустым, так как TEX раскрывает первый знак элемента выравнивания, прежде чем посмотреть на шаблон, чтобы узнать, не начинается ли элемент с `\noalign` или `\omit`. Значение основного счетчика, которое присутствует в начале элемента — это значение в счетчике сразу после того, как "*и* часть" шаблона была полностью прочитана.

строк. Например, следующие тексты прекрасно уравновешиваются, когда ширина первой колонки равна 152.3751 pt, а второй — 171.6249 pt. Но если бы ширина обеих колонок была равна 162 pt, то вторая колонка получилась бы длиннее.

A. The creative part is really more interesting than the deductive part. Instead of concentrating just on finding good answers to questions, it's more important to learn how to find good questions!

B. You've got something there. I wish our teachers would give us problems like, "Find something interesting about x," instead of "Prove x."

A. Exactly. But teachers are so conservative, they'd be afraid of scaring off the "grind" type of students who obediently and mechanically do all the homework. Besides, they wouldn't like the extra work of grading the answers to nondirected questions.

The traditional way is to put off all creative aspects until the last part of graduate school. For seventeen or more years, a student is taught examsmanship, then suddenly after passing enough exams in graduate school he's told to do something original.

A. Творчество на самом деле намного интереснее дедукции. Вместо того, чтобы ограничиться только поиском хороших ответов, важнее научиться ставить хорошие вопросы.

B. В этом вы правы. Я бы хотел, чтобы учителя ставили перед нами вопросы типа "Найдите что-нибудь интересное о x", а не "Докажите $x\ldots$".

A. Именно. Но учителя обычно так консервативны, что будут бояться отпугнуть студентов-"зубрил", которые привыкли послушно и механически выполнять все домашние задания. Кроме того, им может не понравиться дополнительная работа по оценке ответов на нестандартные вопросы.

По традиции творческие аспекты откладываются на старшие курсы. В течение семнадцати или более лет студентов учат сдавать экзамены, а затем вдруг, после успешной сдачи всех экзаменов, предлагают им сделать что-нибудь оригинальное.

Некоторые реализации TEX'а дают изображение по мере работы, чтобы вы могли выбирать ширину колонок интерактивно до тех пор, пока не будет получено подходящее равновесие. Забавно поиграть с такими системами, но можно также попросить TEX вычислять ширину колонок автоматически. Следующая кодировка делает до десяти попыток в поисках решения, когда естественная высота двух колонок различается меньше чем на заданную величину `\delheight`. Предполагается, что макрокоманды `\firstcol` и `\secondcol` генерируют колонки, а сумма ширин колонок равна `\doublewidth`.

```
\newdimen\doublewidth \newdimen\delheight \newif\iffail \newcount\n
\newdimen\trialwidth \newdimen\lowwidth \newdimen\highwidth
\def\balancetwocols{\lowwidth=10em % lower bound on \trialwidth
  \highwidth=\doublewidth \advance\highwidth-10em % upper bound
  {\n=1 \hbadness=10000 \hfuzz=\maxdimen % disable warnings
    \loop \maketrial \testfailure \iffail \preparenewtrial \repeat}
  \maketrial} % now under/overfull boxes will be shown
\def\maketrial{\trialwidth=.5\lowwidth \advance\trialwidth by.5\highwidth
  \setbox0=\vbox{\hsize=\trialwidth \firstcol}
  \setbox2=\vbox{\hsize=\doublewidth\advance\hsize-\trialwidth\secondcol}}
\def\testfailure{\dimen0=\ht0 \advance\dimen0-\ht2
  \ifnum\dimen0<0 \dimen0=-\dimen0 \fi
  \ifdim\dimen0>\delheight \ifnum\n=10 \failfalse\else\failtrue\fi
  \else\failfalse\fi}
\def\preparenewtrial{\ifdim\ht0>\ht2 \global\lowwidth=\trialwidth
```

```
\else\global\highwidth=\trialwidth\fi \advance\n by1 }
```

Ширина каждой из колонок будет не меньше 10 em. Эта программа осуществляет "бинарный поиск", предполагая, что колонки не будут увеличиваться по высоте, когда становятся шире. Если за 10 попыток решение не будет найдено, то желаемого равновесия, по-видимому, достичь невозможно, потому что небольшое увеличение ширины более высокой колонки делает ее короче, чем вторая. Во время установления пробных значений величины `\hbadness` и `\hfuzz` делаются бесконечными, поэтому предупреждающие сообщения, относящиеся к неиспользованным боксам, становятся неуместными. После того, как решение будет найдено, оно вычисляется снова, так что будут выдаваться соответствующие предупреждающие сообщения.

Когда бокс будет помещен в регистр бокса, можно менять его высоту, ширину или глубину, присваивая новые значения `\ht`, `\wd` или `\dp`. Такие присваивания ничего не изменяют внутри бокса, в частности, они не влияют на распределение клея.

Но изменения размеров боксов могут создать путаницу, если вы в точности не представляете, как TEX обращается с боксами в списках. Правила приведены в главе 12, но будет полезно изложить их здесь несколько в другом виде. Если задан бокс и положение его точки привязки, то TEX располагает внутренние боксы следующим образом. (1) Если бокс является h-боксом, то TEX начинает с точки привязки и движется через горизонтальный список внутрь. Когда список содержит бокс, TEX фиксирует точку привязки заключенного бокса в текущем положении и передвигается вправо на ширину бокса. Когда список содержит клей, керн и так далее, TEX перемещается вправо на соответствующую величину. (2) Если бокс является v-боксом, то TEX начинает с верхнего левого угла (то есть TEX сначала передвигается вверх от точки привязки на высоту бокса), а затем движется через вертикальный список внутри. Когда список содержит бокс, TEX помещает правый верхний угол этого бокса в текущее положение, то есть передвигается вниз на высоту этого бокса, затем помещает точку привязки бокса в текущее положение, затем передвигается вниз на глубину бокса. Когда список содержит клей, керн и так далее, TEX перемещается вниз на соответствующую величину.

В соответствии с этими правилами мы можем разобраться в том, что происходит, когда размеры бокса изменяются. Пусть `\delta` будет регистром типа ⟨размер⟩, и пусть `\h` и `\hh` задают горизонтальные списки, которые не зависят от `\box0`. Рассмотрим следующую макрокоманду:

```
\newdimen\temp \newdimen\delta
\def\twohboxes#1{\setbox1=\hbox{\h \copy0 \hh}
  \temp=#10 \advance\temp by \delta #10=\temp
  \setbox2=\hbox{\h \copy0 \hh}}
```

Например, `\twohboxes\wd` создает два горизонтальных бокса, `\box1` и `\box2`, которые идентичны, за исключением того, что ширина `\box0` была увеличена на δ в `\box2`. Какое это имеет значение? Могут быть несколько случаев, в зависимости от того, каким будет `#1`, `\wd`, `\ht` или `\dp`, а также в зависимости от того, будет ли `\box0` горизонтальным или вертикальным боксом. *Случай 1*, `\twohboxes\wd`. Материал из `\hh` сдвигается вправо на δ в `\box2` по сравнению с его положением в `\box1`. Кроме того, `\wd2` на δ больше, чем `\wd1`. *Случай 2*, `\twohboxes\ht`. Если

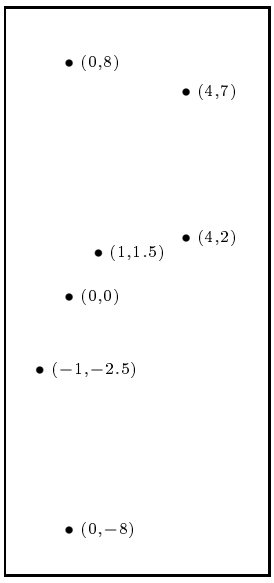
`\box0` является h-боксом, то все остается в том же положении, но если `\box0` является v-боксом, то все в `\copy0` сдвигается вверх на δ. Кроме того, `\ht2` может отличаться от `\ht1`. *Случай 3*, `\twohboxes\dp`. Все остается в том же положении, но `\dp2` может отличаться от `\dp1`.

Аналогично, мы можем разобраться в изменениях, когда изменяются размеры для боксов внутри вертикальных списков. В этом случае мы будем игнорировать влияние междустрочного клея, определив `\twovboxes` следующим образом:

```
\def\twovboxes#1{
  \setbox1=\vbox{\v\nointerlineskip\copy0\nointerlineskip\vv}
  \temp=#10 \advance\temp by \delta #10=\temp
  \setbox2=\vbox{\v\nointerlineskip\copy0\nointerlineskip\vv}}
```

Какая теперь разница между `\box1` и `\box2`? *Случай 1*, `\twovboxes\wd`. Все остается в том же положении, но `\wd2` может отличаться от `\wd1`. *Случай 2*, `\twovboxes\ht`. Если `\box0` является h-боксом, то все в `\v` сдвигается вверх на δ в `\box2` по сравнению с соответствующим положением в `\box1`, если сделать точки привязки двух боксов идентичными; но если `\box0` является v-боксом, то все в нем сдвигается вверх на δ вместе с материалом в `\v`. Кроме того, `\ht2` на δ больше, чем `\ht1`. *Случай 3*, `\twovboxes\dp`. Если `\vv` пусто, то `\dp2` на δ больше, чем `\dp1`, и больше ничего не изменяется. Иначе все в `\v` и в `\copy0` сдвигается вверх на δ, а `\ht2` на δ больше, чем `\ht1`.

Боксы в TEX'е могут объединяться горизонтально и вертикально, но не по диагонали. Но это ограничение можно преодолеть с помощью отрицательных интервалов. Например, семь точек в схеме справа от этого абзаца были набраны очень просто:




```
\hbox{\unit=\baselineskip
  \point 0 0
  \point 0 8
  \point 0 -8
  \point -1 -2.5
  \point 4 7
  \point 4 2
  \point 1 1.5
  }
```

Макрокоманда `\point` создает бокс нулевой ширины, следовательно, отдельные спецификации `\point` могут быть даны в любом порядке, а на координаты не накладываются никакие ограничения:

```
\newdimen\unit
\def\point#1 #2 {\rlap{\kern#1\unit
    \raise#2\unit\hbox{$
      \scriptstyle\bullet\;(#1,#2)$}}}
```

Если спецификации `\point` не заключены в `\hbox` — то есть если они возникают в вертикальной моде — то можно использовать аналогичную конструкцию. В этом

случае \point будет создавать бокс, имеющий нулевую высоту и глубину:

```
\def\point#1 #2 {\vbox to0pt{\kern-#2\unit
    \hbox{\kern#1\unit$\scriptstyle\bullet\;(#1,#2)$}\vss}
  \nointerlineskip}
```

(\nointerlineskip необходимо для того, чтобы клей между строками не путал картинку.)

Если вы дурачитесь, рисуя картинки, а не набираете обычный текст, TEX будет для вас источником бесконечных развлечений и разочарований, потому что если есть подходящие шрифты, вы можете сделать почти все. Предположим, например, что у вас есть шрифт \qc, который содержит четыре четверти окружности

a = b = c = d =

Эти символы имеют одинаковую высоту, ширину и глубину. Ширина и высота плюс глубина равны диаметру соответствующей полной окружности. Кроме того, точка привязки каждого символа находится в особом месте. Каждая четверть дуги имеет такую горизонтальную конечную точку, что левый край кривой находится на базовой линии, а вертикальная конечная точка такова, что левый край находится прямо над или под точкой привязки. Это позволяет гарантировать полное согласование между этими символами и прямыми, которые встречаются с ними в конечных точках. Толщина таких прямых должна быть \fontdimen8\qc.

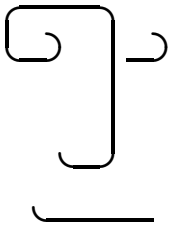
Если заданы такие символы, можно придумать макрокоманды \path, \L, \R, \S и \T такие, что \path{⟨любая строка из \L, \R, \S, и \T⟩} дает траекторию, которая начинается с путешествия на восток, затем поворачивает влево для каждого \L, вправо для каждого \R, идет прямо для каждого \S и поворачивает назад для каждого \T. Таким образом, например, \path{\L\T\S\T\R\L\T\S\T\R} дает а также можно получить следующие эффекты:

```
\path{\L\R\S\R\S\R\S\S\R\R}
```


```
\path{\R\R\R\R\T\S\S\L\L\L\L\L\S\S}
```


```
\def\X{\L\T\L\L\T\L\L\T} \path{\X\X\X\X}
```


Кроме того, имеются операции \B и \W, которые, соответственно, делают траекторию черной (видимой) и белой (невидимой):

```
\path{\R\R\S
  \W\S\S\S\R\R
  \B\R\R\S\R\S\R\S\S\S\R\S\S\S\S\S\R\S\R
  \W\R\R\R\S\L\S
  \B\L\S\S\S\S}
```


(Может быть понадобится поместить керны перед и после траектории, так как бокс, полученный макрокомандой \path, может иметь не такую ширину, как сама траектория.)

Макрокоманды \path действуют не так, как \point, так как боксам здесь не нужно иметь нулевую ширину:

```
\catcode`\ =9 \endlinechar=-1 % игнорировать все пробелы (временно)
\newcount\dir \newdimen\y \newdimen\w
\newif\ifvisible \let\B=\visibletrue \let\W=\visiblefalse
\newbox\NE \newbox\NW \newbox\SE \newbox\SW \newbox\NS \newbox\EW
\setbox\SW=\hbox{\qc a} \setbox\NW=\hbox{\qc b}
\setbox\NE=\hbox{\qc c} \setbox\SE=\hbox{\qc d}
\w=\wd\SW \dimen0=\fontdimen8\qc
\setbox\EW=\hbox{\kern-\dp\SW \vrule height\dimen0 width\wd\SW} \wd\EW=\w
\setbox\NS=\hbox{\vrule height\ht\SW depth\dp\SW width\dimen0}  \wd\NS=\w
\def\L{\ifcase\dir \dy+\NW \or\dx-\SW \or\dy-\SE \or\dx+\NE\dd-4\fi \dd+1}
\def\S{\ifcase\dir \dx+\EW \or \dy+\NS \or \dx-\EW \or \dy-\NS \fi}
\def\R{\ifcase\dir \dy-\SW\dd+4 \or\dx+\SE \or\dy+\NE \or\dx-\NW\fi \dd-1}
\def\T{\ifcase\dir\kern-\w\dd+2\or\ey-\dd+2\or\kern\w\dd-2\or\ey+\dd-2\fi}
\edef\dd#1#2{\global\advance\dir#1#2\space}
\def\dx#1#2{\ifvisible\raise\y\copy#2 \if#1-\kern-2\w\fi\else\kern#1\w\fi}
\def\dy#1#2{\ifvisible\raise\y\copy#2 \kern-\w \fi \global\advance\y#1\w}
\def\ey#1{\global\advance\y#1\w}
\def\path#1{\hbox{\B \dir=0 \y=0pt #1}}
\catcode`\ =10 \endlinechar=`\^^M % восстановлены нормальные соглашения

\newcount\n % the current order in the \dragon and \nogard macros
\def\dragon{\ifnum\n>0{\advance\n-1 \dragon\L\nogard}\fi}
\def\nogard{\ifnum\n>0{\advance\n-1 \dragon\R\nogard}\fi}
```

(Три последние строки не являются частью макрокоманды \path, но их можно использовать как интересный тест. Чтобы получить знаменитую "драконовскую кривую", порядка 9, нужно только написать \path{\dir=3 \n=9 \dragon}.)

А теперь давайте обратимся к другой задаче, ориентированной на боксы. Макрокоманда, \listing, которая обсуждалась в этой главе раньше, была ограничена распечаткой файлов, которые содержат только видимые символы ASCII. Но иногда бывают также нужны знаки табуляции ⟨tab⟩ ASCII, где ⟨tab⟩ эквивалентно 1, 2, $\cdots$ или 8 пробелам (пробелов столько, сколько необходимо, чтобы сделать длину текущей строки кратной 8). Как это можно сделать?

Предположим, что файлы содержат специальный символ, который TEX будет вводить как символ номер 9, код ASCII ⟨tab⟩ (в некоторых реализациях это сделать невозможно). Если файл содержит три символа ^^I, то начальный TEX будет обычно вводить их как один символ номер 9. Но в дословной распечатке файла мы, естественно, хотим, чтобы такие символы печатались такими, какие они есть, то есть как ^^I.

Следующая конструкция переопределяет \setupverbatim так, что предыдущая макрокоманда \listing будет работать с символами ⟨tab⟩. Идея состоит в том, чтобы держать уже появившуюся часть строки в h-боксе, который можно

“измерить”, чтобы выяснить, сколько символов появилось после начала строки или после последнего ⟨tab⟩.

```
\def\setupverbatim{\tt \lineno=0
  \def\par{\leavevmode\egroup\box0\endgraf}
  \obeylines \uncatcodespecials \obeyspaces
  \catcode`\`=\active \catcode`\^^I=\active
  \everypar{\advance\lineno by1
    \llap{\sevenrm\the\lineno\ \ }\startbox}}
\newdimen\w \setbox0=\hbox{\tt\space} \w=8\wd0 % величина tab
\def\startbox{\setbox0=\hbox\bgroup}
{\catcode`\^^I=\active
  \gdef^^I{\leavevmode\egroup
    \dimen0=\wd0 % ширина от начала или от предыдущего tab
    \divide\dimen0 by\w
    \multiply\dimen0 by\w % вычисление предыдущего произведения \w
    \advance\dimen0 by\w  % добавлено следующее произведение \w
    \wd0=\dimen0 \box0 \startbox}}
```

(Новым в этом определении `\setupverbatim` является `\egroup\box0` в переопределении `\par`, `\catcode`\^^I=\active` и `\startbox` в `\everypar`.) Макрокоманды `\settabs` и `\+` приложения B дают другой пример того, как операции табулирования могут быть смоделированы с помощью боксирования и разбоксирования.

В главе 22 объясняется, как помещать вертикальные прямые в таблицы, рассматривая прямые как отдельные колонки. Но есть и другой способ, при условии, что прямые в таблице на всем протяжении проходят сверху донизу. Например,

```
\beginvrulealign
\tabskip=10pt
\halign{&\strut#\hfil\cr
Эти&    после того,\cr
вертикальные& как\cr
прямые&  таблица\cr
были&     была\cr
вставлены& закончена!\cr}
\endvrulealign
```

дает

Эти	после того,
вертикальные	как
прямые	таблица
были	была
вставлены	закончена!

В этом случае магические макрокоманды проверяют нижнюю строку выравнивания , которая состоит из чередующихся табличного клея и боксов. Каждый элемент табличного клея в этой нижней строке будет разделен пополам вертикальной прямой. Это происходит так:

```
\def\beginvrulealign{\setbox0=\vbox\bgroup}
\def\endvrulealign{\egroup % теперь \box0 хранит все выравнивание
  \setbox0=\vbox{\setbox2=\hbox{\vrule height\ht0 depth\dp0 width0pt}
    \unvbox0 \setbox0=\lastbox % теперь \box0 является нижней строкой
    \nointerlineskip \copy0 % поместить это обратно
    \global\setbox1=\hbox{} % инициализировать бокс, содержащий прямые
    \setbox4=\hbox{\unhbox0 % открыть нижнюю строку
```

```
    \loop \skip0=\lastskip \unskip % удалить  табличный клей
     \advance\skip0 by-.4pt % прямые имеют ширину  .4pt
     \divide\skip0 by 2
     \global\setbox1=\hbox{\hskip\skip0\vrule\hskip\skip0
       \unhbox2\unhbox1}%
     \setbox2=\lastbox % удалить ввод выравнивания
     \ifhbox2 \setbox2=\hbox{\kern\wd2}\repeat}%
\hbox{\rlap{\box0}\box1}} % наложить выравнивание на прямые
```

Этот метод годится для всех выравниваний, созданных командами `\halign{...}`. Для выравниваний, созданных, скажем, `\halign to100pt{...}`, этот метод подходит только когда нижняя строка выравнивания содержит все столбцы и когда `\box1` заменяется на `\hbox to100pt{\unhbox1}` в конце `\endvrulealign`.

7. Маневры с абзацами. В главе 14 было обещано, что в приложении D будет дан пример, где в одном абзаце встречаются неровный правый край и неровный левый край. Следующий интересный пример был предложен в "Key Index" в *Mathematical Reviews*, где элементы состоят из как можно более длинного заголовка, за которыми следуют точечные проводники, а затем как можно более длинный список номеров обзоров. Если заголовок не помещается на одной строке, он должен быть задан неровным справа с подвешенным отступом всех строк после первой. Если ссылки не помещаются на одной строке, они должны быть заданы неровными слева. Например, если на входе задано

```
ACM Symposium on Theory of Computing, Eighth Annual (Hershey, %
Pa., 1976)\:1879, 4813, 5414, 6918, 6936, 6937, 6946, 6951, %
6970, 7619, 9605, 10148, 11676, 11687, 11692, 11710, 13869
```

то в зависимости от ширины колонок могут понадобиться следующие три типа результата:

ACM Symposium on
Theory of Computing,
Eighth Annual
(Hershey, Pa., 1976)
..... 1879, 4813, 5414,
6918, 6936, 6937, 6946,
6951, 6970, 7619, 9605,
10148, 11676, 11687,
11692, 11710, 13869

ACM Symposium on Theory of
Computing, Eighth Annual
(Hershey, Pa., 1976) .....
1879, 4813, 5414, 6918, 6936, 6937,
6946, 6951, 6970, 7619, 9605, 10148,
11676, 11687, 11692, 11710, 13869

ACM Symposium on Theory of Computing, Eighth Annual
(Hershey, Pa., 1976) ... 1879, 4813, 5414, 6918, 6936, 6937, 6946,
6951, 6970, 7619, 9605, 10148, 11676, 11687, 11692, 11710, 13869

Отметим, что точечные проводники трактуются тремя различными способами в зависимости от того, какой больше подходит: они могут появляться на первой строке слева после заголовка, либо находиться в конце последней строки заголовка (в этом случае они заканчиваются как раз перед правым полем), либо находиться в середине строки. Кроме того, предполагается, что строки, неровные справа, заканчиваются не меньше чем за 0.5 em от правого поля. Наша цель состоит в том, чтобы получить это как частный случай общего метода построения абзацев TEX'а. Простой подход из приложения B не подходит, потому что `\raggedright`

получается здесь с помощью регулировки `\rightskip`. TEX использует одно и то же значение `\rightskip` во всех строках абзаца.

Решение этой задачи требует понимания алгоритма разбиения строк. Оно зависит от того, как вычисляется дефектность и как удаляются элементы в точке разрыва, так что читателю следует еще раз просмотреть главу 14, чтобы полностью разобраться в этих понятиях. По существу, нужно задать последовательность элементов бокс/клей/штраф для промежутков в заголовке, другую последовательность для промежутков в ссылках и еще одну последовательность для точечных проводников. В заголовке для каждого элемента индексов промежутки между словами могут быть представлены последовательностью

```
\penalty10000 \hskip.5em plus3em \penalty0
\hskip-.17em plus-3em minus.11em
```

Таким образом, имеется растяжимость на 3 em, если разрыв строки происходит на `\penalty0`. В противном случае промежуток между словами будет равен .33 em и сжиматься до .22 em. Это дает неровное правое поле. Промежутки между словами в ссылках рассчитаны таким образом, чтобы получилось неровное левое поле и чтобы количество строк, посвященных ссылкам, сводилось к минимуму:

```
\penalty1000 \hskip.33em plus-3em minus.11em
\vadjust{}\penalty10000 \hskip0pt plus3em
```

`\vadjust{}` не делает ничего, но оно не исчезает при разрыве строки. Таким образом, если разрыв происходит на `\penalty1000`, следующая строка будет начинаться с растяжимости 3 em, но если разрыва не происходит, то промежуток будет равен .33 em минус .11 em. И, наконец, переход между заголовком и ссылками может быть задан с помощью

```
\penalty10000 \hskip.5em plus3em \penalty600
\hskip+.17em plus-3em minus.11em
\vadjust{}\penalty10000
\leaders\copy\dbox\hskip-3.3\wd\dbox plus1fil minus.3\wd\dbox
\kern3em \penalty600 \hskip-2.67em plus-3em minus.11em
\vadjust{}\penalty10000 \hskip0pt plus3em
```

(Прожевали?) Эта длинная последовательность штрафов и клеев начинается весьма похоже на промежутки между словами в первой части, а заканчивается весьма похоже на промежутки между словами в последней части. Она имеет две разрешенные точки разрыва, а именно, на `\penalty600`. Первая точка разрыва ведет к появлению точечных проводников в начале строки. Вторая ведет к появлению их в конце, но на расстоянии 3 em. Ширина точечных проводников всегда будет по меньшей мере равна ширине `\dbox`, умноженной на три, так что всегда будет появляться по меньшей мере две копии `\dbox`. Приведем реальную кодировку TEX'а, которую можно использовать, чтобы задать желаемое поведение:

```
\hyphenpenalty10000 \exhyphenpenalty10000 \pretolerance10000 % дефисов нет
\newbox\dbox \setbox\dbox=\hbox to .4em{\hss.\hss} % бокс для проводников
\newskip\rrskipb \rrskipb=.5em plus3em % неровный пробел справа до разрыва
\newskip\rrskipa \rrskipa=-.17em plus-3em minus.11em % то же, после него
\newskip\rlskipa \rlskipa=0pt plus3em % неровный пробел слева после разрыва
```

```
\newskip\rlskipb \rlskipb=.33em plus-3em minus .11em % то же, перед ним
\newskip\lskip \lskip=3.3\wd\dbox plus1fil minus.3\wd\dbox % для многоточия
\newskip\lskipa \lskipa=-2.67em plus-3em minus.11em % после многоточия
\mathchardef\rlpen=1000 \mathchardef\leadpen=600 % константы
\def\rrspace{\nobreak\hskip\rrskipb\penalty0\hskip\rrskipa}
\def\rlspace{\penalty\rlpen\hskip\rlskipb\vadjust{}\nobreak\hskip\rlskipa}
\uccode`~=` \uppercase{
  \def\:{\nobreak\hskip\rrskipb \penalty\leadpen \hskip\rrskipa
    \vadjust{}\nobreak\leaders\copy\dbox\hskip\lskip
    \kern3em \penalty\leadpen \hskip\lskipa
    \vadjust{}\nobreak\hskip\rlskipa \let~=\rlspace}
  \everypar{\hangindent=1.5em \hangafter=1 \let~=\rrspace}}
\uccode`~=0 \parindent=0pt \parfillskip=0pt \obeyspaces
```

Помещение клея между словами в регистры `\skip` позволяет экономить время и память, когда TEX работает с такими абзацами: `\hskip`⟨явный клей⟩ занимает шесть ячеек памяти боксов TEX'a , а `\hskip`⟨skip-регистр⟩ занимает только две. Отметим здесь искусное использование `\uppercase`, чтобы преобразовать ~$_{13}$ в ␣$_{13}$. Аналогично можно получить "случайные" активные символы.

Давайте теперь обратимся к более простой задаче: к *подвешенной пунктуации*

"Что такое подвешенная пунктуация?" — спросила Алиса, вопрсительно приподняв брови. 'Посмотри сама,' — ответил Билл, — 'это легче показать, чем объяснить.' "О, теперь я понимаю. Запятые, точки и кавычки могут вылезать на поля, если они попадают на разрыв строки." 'Да, похоже." "Понятно! Но почему твои слова стоят в простых кавычках, а мои в двойных?" 'Не имею не малейшего представления, это действительно странно. Спроси у автора этой сумасшедшей книги.'

Каждая запятая в абзаце об Алисе и Билле была представлена внутри TEX'a последовательностью "`,\kern-\commahang\kern\commahang`". Аналогичные замены были сделаны для точек и для закрывающих кавычек. Открывающие кавычки были представлены более длинной последовательностью

```
\kern\qquotehang\vadjust{}\kern-\qquotehang``\allowhyphens
```

где `\allowhyphens` позволяет переносить следующее слово. Эта конструкция работает, потому что керны исчезают на разрывах строк должным образом. Соответствующие правила из главы 14 таковы. (1) Разрыв строки может произойти только на керне, сразу за которым следует клей. (2) Последовательные клей, керн и штраф исчезают при разрыве строки.

Чтобы задать TEX'у подвешенную пунктуацию, можно указать

```
\newdimen\commahang      \setbox0=\hbox{,}      \commahang=\wd0
\newdimen\periodhang     \setbox0=\hbox{.}      \periodhang=\wd0
\newdimen\quotehang      \setbox0=\hbox{`}      \quotehang=\wd0
\newdimen\qquotehang     \setbox0=\hbox{``}     \qquotehang=\wd0
\newskip\zzz \def\allowhyphens{\nobreak\hskip\zzz}
\def\lqq{``} \def\rqq{''} \def\pnt{.}
\def\comma{,\kern-\commahang\kern\commahang}
```

```
\def\period{.\kern-\periodhang\kern\periodhang}
\def\rquote{'\kern-\quotehang\kern\quotehang}
\def\lquote{\ifhmode\kern\quotehang\vadjust{}\else\leavevmode\fi
  \kern-\quotehang`\allowhyphens}
\catcode`,=\active \let,=\comma \catcode`.=\active \let.=\period
\catcode`'=\active \def'{\futurelet\next\rqtest}
\catcode``=\active \def`{\futurelet\next\lqtest}
\def\rqtest{\ifx\next'\let\next=\rquotes\else\let\next=\rquote\fi\next}
\def\lqtest{\ifx\next`\let\next=\lquotes\else\let\next=\lquote\fi\next}
\def\rquotes'{\rqq\kern-\qquotehang\kern\qquotehang}
\def\lquotes`{\ifhmode\kern\qquotehang\vadjust{}\else\leavevmode\fi
  \kern-\qquotehang\lqq\allowhyphens}
```

Отметим, что макрокомандам необходимо выполнить свои собственные проверки на лигатуры. Они также предпринимают соответствующие действия, когда абзац начинается с открывающих кавычек. Так как `\kern` не влияет на коэффициент пробелов, то подвешенная пунктуация не влияет на соглашения TEX'а о распределении пробелов внутри строки. Частично подвешенная пунктуация может быть получена путем уменьшения значения величин `\commahang` и тому подобных. Макрокоманды `\pnt`, `\lq` и `\rq` следует использовать и в константах. Например, размер 6.5 дюймов, когда действует подвешенная пунктуация, должен быть записан как `6\pnt5in`, а команда `\catcode\lq,=12` делает запятые снова неактивными символами. Для получения "подвешенного переноса" следует использовать специальный шрифт с `\hyphenchar` нулевой ширины.

А теперь рассмотрим несколько приемов, с помощью которых можно организовать размещение на странице коротких сносок. Сноски, расположенные внизу этой страницы[1,2,3,4,5,6,7,8,9,10] выглядят не очень симпатично, потому что большинство из них очень короткие. Когда в документе встречается множество сносок и когда большинство из них занимает только незначительную часть строки, программе вывода следует переформатировать их более подходящим образом.

[1] Первая сноска.

[2] Вторая сноска. (Появляющаяся время от времени длинная сноска сильно усложняет дело.)

[3] Третья сноска.

[4] Четвертая сноска.

[5] Пятая сноска (Это очень скучно, но это для примера.)

[6] Еще одна.

[7] И еще одна.

[8] И еще.

[9] Бесконечная сноска.

[10] Множество их.

Один из возможных подходов, например, состоит в том, чтобы набирать сноски узкими колонками и поместить, скажем, три колонки сносок внизу каждой страницы. Десять сносок, приведенных в примере, могут выглядеть так:

[1] Первая сноска.
[2] Вторая сноска. (Появляющаяся время от времени длинная сноска сильно усложняет дело.)
[3] Третья сноска.
[4] Четвертая сноска.
[5] Пятая сноска. (Это очень скучно, но это для примера.)
[6] Еще одна.
[7] И еще.
[8] И еще.
[9] Бесконечная сноска.
[10] Множество их.

В этом случае сноски могут генерироваться посредством

```
\insert\footins{\eightpoint \hsize=9pc \parindent=1pc
  \leftskip=0pt \raggedright \pretolerance=10000
  \hyphenpenalty=10000 \exhyphenpenalty=10000
  \interlinepenalty=\interfootnotelinepenalty
  \floatingpenalty=20000
  \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
  \item{$^{\the\footno}$}\strut⟨text of footnote⟩\strut
  \par\allowbreak}
```

а `\count\footins` должен быть установлен в 333, с тем чтобы каждая строка сноски занимала примерно одну треть страницы. Программа вывода тогда встретит `\box\footins`, который имеет такой вид:

```
\vbox(142.0+2.0)x108.0
.\hbox(7.0+2.0)x108.0, glue set 38.27864fil []
.\penalty 0
.\hbox(7.0+2.0)x108.0, glue set 2.23752 []
.\penalty 250
.\glue(\baselineskip) 1.0
.\hbox(6.0+2.0)x96.0, glue set 2.05025, shifted 12.0 []
.\penalty 100
.\glue(\baselineskip) 3.55556
.\hbox(3.44444+1.55556)x96.0, glue set 1.5984, shifted 12.0 []
   ⋮
.\hbox(7.0+2.0)x108.0, glue set 17.73251fil []
.\penalty 0
.\hbox(7.0+2.0)x108.0, glue set 38.04253fil []
.\penalty 0
```

В конце каждой строки стоит `\penalty 0`. Сноски, которые занимают больше одной строки, имеют большие штрафы между строками, а также здесь появляется и междустрочный клей.

Как будет программа вывода разделять такой бокс на три примерно равные части? Отметим, что содержимое бокса является полностью неизменным, то есть в нем нет клея, который может растягиваться или сжиматься. Кроме того, можно предположить, что содержимое бокса является постоянным, то есть расстояния между базовыми линиями одинаковы. При таких условиях для того чтобы

разделить бокс на три равные части, можно использовать довольно простую балансирующую программу.

Давайте рассмотрим более общую задачу. Предположим, что дан жесткий v-бокс высотой в n строк, в котором соседние базовые линии находятся на расстоянии b единиц. Предположим также, что верхняя базовая линия находится на расстоянии h единиц от вершины v-бокса, где $0 < h < b$. (В нашем примере со сносками $b = 9\,\mathrm{pt}$ и $h = 7\,\mathrm{pt}$. В стандартах начального TEX'а $b = 12\,\mathrm{pt}$ и $h = 10\,\mathrm{pt}$. Мы могли бы также решить задачу для обобщенных b и h.) Отсюда следует, что высота v-бокса равна $H = h + b(n-1) = bn + h - b$.

Если n строк нужно равномерно распределить в k колонок, то первая колонка будет содержать $\lceil n/k \rceil$ строк. (Это обозначает наименьшее целое, которое больше или равно n/k.) Например, в нашем примере со сносками $n = 16$ и $k = 3$, следовательно, первая колонка будет содержать 6 строк. После формирования первой колонки мы свели задачу к $n = 10$ и $k = 2$, так что операция закончится двумя колонками по 5 строк в каждой. (Отметим, что лучше разделить 16 на $6+5+5$, а не на $6+6+4$.) Когда мы найдем первую колонку, всегда можно будет свести задачу с k колонками к задаче с $(k-1)$ колонками, так что нам нужно сосредоточиться на нахождении первой колонки.

Пусть $m = \lceil n/k \rceil$. Высота данного бокса равна $bn + h - b$, а высота первой колонки должна быть $bm + h - b$, следовательно, мы хотим сделать `\vsplit` этой высоты. Нам, однако, не нужно точно вычислять $bm + h - b$, т.к. простые арифметические действия доказывают, что

$$bm + h - b \;<\; \frac{bn + h - b}{k} + h \;<\; b(m+1) + h - b.$$

Поэтому достаточно придать `\vsplit` высоту $H' = H/k + h$. Предполагая жесткость и что разрыв возможен после каждой строки, `\vsplit` с H' произойдет после максимального количества строк, которые дают бокс высотой $\le H'$. (Мы заметили, что m строк дают бокс высотой $< H'$, в то время как $m+1$ строк дают бокс высотой $> H'$.) Это делается с помощью следующей программы:

```
\newcount\k \newdimen\h % регистры, используемые программой \rigidbalance
\def\rigidbalance#1#2 #3 {\setbox0=\box#1 \k=#2 \h=#3
  \line{\splittopskip=\h \vbadness=10000 \hfilneg
    \valign{##\vfil\cr\dosplits}}}
\def\dosplits{\ifnum\k>0 \noalign{\hfil}\splitoff
  \global\advance\k-1\cr\dosplits\fi}
\def\splitoff{\dimen0=\ht0
  \divide\dimen0 by\k \advance\dimen0 by\h
  \vsplit0 to \dimen0 }
```

Эта программа интересна по ряду причин. Во-первых, отметим, что вычисление зависит не от b, а только от h и высоты данного бокса. Следовательно, `\rigidbalance` имеет три параметра: номер регистра бокса, количество колонок k и высоту верхней базовой линии h. Программа разделяет данный v-бокс на k примерно равных частей и вставляет результат в `\line`. Значение `\splittopskip` устанавливается в h, так что последующие v-боксы будут удовлетворять основным правилам начального v-бокса и задача сводится от k к $k-1$. Каждой колонке будет

предшествовать \hfil, следовательно, \hfilneg используется для того, чтобы удалить \hfil перед первой колонкой. \valign используется для выравнивания всех колонок сверху. Отметим, что преамбула к этому \valign очень проста, а тело \valign генерируется рекурсивной макрокомандой \dosplits, которая производит k колонок. Значение \vbadness устанавливается в 10000, потому что каждая операция \vsplit будет производить недозаполненный v-бокс, плохость которого равна 10000.

В нашем примере со сносками программа вывода может переформатировать содержимое \box\footins, например, таким образом:

```
\rigidbalance\footins 3 7pt
\setbox\footins=\lastbox
```

так что \lastbox будет результатом \rigidbalance.

Такое решение задачи коротких сносок может привести к появлению висячих строк, так как описанная нами балансирующая программа просто делит общее количество строк на три равные части. Например, если бы десятая сноска из нашего примера отсутствовала, то оставшиеся 15 строк были бы разбиты так: $5 + 5 + 5$. Во второй колонке тогда были бы слова "усложняет дело.)", а третья колонка начиналась бы с "для примера.)". Жесткую процедуру балансировки можно было бы заменить другой, допускающей колонки, неровные снизу, но есть и другой подход: весь набор сносок можно было бы объединить в один абзац с большими пробелами между отдельными элементами. Например, десять рассмотренных нами сносок могли бы иметь такой вид:

[1]Первая сноска. [2]Вторая сноска. (Появляющаяся время от времени длинная сноска сильно усложняет дело.) [3]Третья сноска. [4]Четвертая сноска. [5]Пятая сноска. (Это очень скучно, но это для примера.) [6]Еще одна. [7]И еще одна. [8]И еще. [9]Бесконечная сноска. [10]Множество сносок.

Можно было бы взять показанное ранее содержимое \box\footins и переформатировать все в абзац, но такая операция была бы неоправданно сложной. Если сноски должны быть организованы в абзац программой вывода, то лучше всего просто подготовить их в невыровненных h-боксах. Каждый из этих h-боксов будет позже разбоксирован, так что мы можем свободно обращаться с их высотой, шириной и глубиной. Удобно установить глубину равной нулю, а высоту — отметке, соответствующей тому, какой вклад конкретная сноска внесет в результирующий абзац. Например, если сноска занимает точно половину \hsize и если конечную сноску собираются установить с \baselineskip=10pt, то высота h-бокса сноски будет установлена равной 5 pt. Допустив, что \count\footins=1000, мы получим весьма хорошую оценку размера абзаца последней сноски. Иными словами, предлагается следующая схема вставки:

```
\insert\footins{\floatingpenalty=20000
  \eightpoint \setbox0=\hbox{%
    $^{\the\footno}$⟨text of footnote⟩\penalty-10\hskip\footglue}
  \dp0=0pt \ht0=\fudgefactor\wd0 \box0}
```

штраф -10 стремится способствовать разрыву строк между сносками, \footglue — это величина клея между сносками в последнем абзаце сноски, а \fudgefactor

— это отношение `\baselineskip` к `\hsize` в этом абзаце. Автор в своих экспериментах определил необходимые величины следующим образом:

```
\eightpoint \newskip\footglue \footglue=1.5em plus.3em minus.3em
\newdimen\footnotebaselineskip \footnotebaselineskip=10pt
\dimen0=\footnotebaselineskip \multiply\dimen0 by 1024
\divide \dimen0 by \hsize \multiply\dimen0 by 64
\xdef\fudgefactor{\expandafter\getfactor\the\dimen0 }
```

(Вычисление `\fudgefactor` использует тот факт, что $1\,\mathrm{pt} = 1024 \times 64\,\mathrm{sp}$. Предполагается также, что `\footnotebaselineskip` меньше, чем 16 pt.)

Внутри программы вывода `\box\footins` будет равен вертикальному боксу, состоящему из h-боксов, а высота этого вертикального бокса будет отметкой высоты последнего абзаца. Например, наши десять сносок дают

```
\vbox(33.72627+0.0)x372.37468
.\hbox(2.11377+0.0)x74.63846 []
.\hbox(10.54576+0.0)x372.37468 []
.\hbox(2.12738+0.0)x75.11902 []
.\hbox(2.4849+0.0)x87.74281 []
.\hbox(6.48363+0.0)x228.93929 []
.\hbox(1.59232+0.0)x56.2257 []
.\hbox(1.79616+0.0)x63.42302 []
.\hbox(1.21774+0.0)x42.99911 []
.\hbox(2.69565+0.0)x95.18459 []
.\hbox(2.66898+0.0)x94.2428 []
```

а высота 33.72627 pt соответствует отметке примерно трех с половиной строк. (Планировщик страниц TEX'а благодаря сноскам добавил также `\skip\footins` при оценке общего вклада.)

Переформатирование `\box\footins` происходит в три этапа. Сначала вертикальный бокс, состоящий из h-боксов, заменяется на горизонтальный бокс, состоящий из h-боксов, так что мы получаем, например,

```
\hbox(10.54576+0.0)x1190.88947
.\hbox(2.11377+0.0)x74.63846 []
    ⋮
.\hbox(2.66898+0.0)x94.2428 []
```

(содержимое такое же, как и прежде, но растянутое в горизонтальную строку, а не в вертикальную колонку). Затем внутренние горизонтальные боксы разбоксируются, и мы получаем

```
\hbox(6.68999+2.0)x1190.88947
.\mathon
.\hbox(3.86665+0.0)x4.16661, shifted -2.82333 []
.\mathoff
.\eightrm Π
.etc.
```

И, наконец, внешний горизонтальный бокс разбоксируется и горизонтальный список внутри него превращается в абзац. Приведем реальную программу TEX'а:

```
\def\makefootnoteparagraph{\unvbox\footins \makehboxofhboxes
  \setbox0=\hbox{\unhbox0 \removehboxes}
  \baselineskip=\footnotebaselineskip\noindent\unhbox0\par}
\def\makehboxofhboxes{\setbox0=\hbox{}
  \loop\setbox2=\lastbox \ifhbox2 \setbox0=\hbox{\box2\unhbox0}\repeat}
\def\removehboxes{\setbox0=\lastbox
  \ifhbox0{\removehboxes}\unhbox0 \fi}
```

Операция `\removehboxes` заслуживает особого внимания, потому что она использует стэк спасения TEX'а для хранения всех h-боксов перед разбоксированием. Каждый уровень рекурсии этой программы использует одну ячейку памяти входного стэка и три ячейки памяти стэка спасения. Таким образом, можно совершенно спокойно делать 100 и более сносок, не переполняя объем памяти TEX'а. Программа `\makehboxofhboxes` не столь эффективна. TEX не позволят разбоксировать вертикальный бокс в горизонтальной моде или наоборот, следовательно, прием с `\removehboxes` не может быть использован. Это означает, что время работы пропорционально n^2, если имеется n сносок, потому что время для создания и аннулирования бокса пропорционально количеству элементов в списке верхнего уровня внутри его. Однако коэффициент пропорциональности мал, так что нет необходимости обращаться к более сложной схеме, которая была бы чуть более быстрее. И в самом деле, сама операция `\lastbox` имеет время работы, равное примерно $a + mb$, где m — количество элементов в списке, предшествующем удаляемому боксу. Следовательно, `\removehboxes` тоже имеет время работы порядка n^2. Но константа b настолько мала, что для практических целей можно считать, что `\lastbox` работает почти мгновенно. Отметим, однако, что было бы ошибкой обходить операцию `\removehboxes`, написав `\setbox0=\hbox{\unhbox2\unhbox0}` в `\makehboxofhboxes`. Это сделало бы список верхнего уровня внутри `\box0` слишком длинным для эффективного разбоксирования.

8. Связь с программами вывода. Можно было бы написать целую книгу о программах вывода TEX'а, но это приложение и так уже слишком длинное, так что ограничимся тем, что укажем только 1–2 тайных приема, о которых трудно догадаться самому. (В приложении E дается еще несколько более простых примеров.)

Иногда программе вывода нужно знать, почему она вызвана, так что возникает проблема передачи информации из остальной части программы. У TEX'а есть общие операции `\mark`, но метки не всегда дают правильные типы ключей. Затем имеется `\outputpenalty`, которую можно тестировать, чтобы посмотреть, какой штраф был в точке разрыва. Любой штраф -10000, -10001, -10002 или меньше вызывает программу вывода, поэтому для передачи различных сообщений можно использовать различные штрафы. (Когда программа вывода помещает материал обратно в список вкладов, ей не нужно восстанавливать штраф в точке разрыва.) Если вывод форсировался высоким отрицательным значением `\outputpenalty`, то программа вывода может использовать `\vbox{\unvcopy255}`, чтобы определить, насколько полной в действительности является страница до этого места. О недозаполненных и переполненных боксах не сообщается, когда `\box255` упаковывается для программы вывода, так что преждевременное выбрасывание страницы не наносит ущерба, если вы хотите передать сигнал.

Пожалуй, самый хитрый из всех приемов — это связь с программой вывода через *глубину* `\box255`. Предположим, например, что вы хотите узнать, кончается ли текущая страница последней строкой абзаца. Если каждый абзац заканчивается `\specialstrut`, где `\specialstrut` аналогична `\strut`, но на 1 sp глубже, то `\dp255` будет иметь распознаваемое значение, если страница заканчивается одновременно с абзацем. (Конечно, `\maxdepth` должно быть достаточно большим. Начальный TEX считает `\maxdepth=4pt`, в то время как подпорки обычно имеют глубину 3.5 pt, поэтому здесь нет проблем.) Расстояние, равное 1000 sp, невидимо невооруженным глазом, так что таким образом можно передавать различные сообщения.

Если значение `\vsize` очень мало, TEX будет строить абзацы как обычно, но посылать их в программу вывода по одной строке. Таким способом программа вывода может присоединять пометки на полях и тому подобное, основываясь на том, что происходит на строке. Абзацы, которые были восстановлены таким способом, могут также передаваться обратно из программы вывода планировщику страниц. Затем будут найдены обычные разрывы страниц, если значение `\vsize` было восстановлено.

Программа вывода может также писать пометки в файл, основываясь на том, что имеется в рукописи. Может быть разработана двухцикловая система, где TEX во время первого цикла только собирает информацию. Реальный набор текста может производиться во время второго цикла с использованием `\read` для восстановления информации, которая была записана во время первого цикла.

9. Проверка синтаксиса. Предположим, что вы хотите пропустить рукопись через TEX просто для того, чтобы проверить ошибки, не получая никакого вывода. Есть ли способ заставить TEX, делая это, работать значительно быстрее? Есть, и вот какой. (1) Написать `\font\dummy=dummy`. Система должна подключать файл `dummy.tfm`, который определяет шрифт без символов (но имеющий достаточно параметров `\fontdimen`, чтобы квалифицировать его как шрифт математических символов). (2) Установить все идентификаторы шрифта, которые вы используете, равными `\dummy`. Например, `\let\tenrm=\dummy`, `\let\tenbf=\dummy`, ..., `\textfont0=\dummy`, и так далее. (3) Написать `\dummy`, чтобы выбрать фиктивный шрифт (поскольку начальный TEX мог выбрать реальный `\tenrm`). (4) Установить `\tracinglostchars=0`, так что TEX не будет жаловаться, когда символы в фиктивном шрифте будут отсутствовать. (5) Установить

```
\output={\setbox0=\box255\deadcycles=0}
```

так что ничего выдаваться не будет, но TEX не будет думать, что ваша программа вывода испорчена. (6) Написать `\newtoks\output`, так что никакая другая программа вывода не будет определена. (7) Написать `\frenchspacing`, так что TEX'у не нужно будет делать вычисления коэффициента пробелов. (8) Написать `\hbadness=10000`, так что о недозаполненных боксах сообщаться не будет. (9) А если вы хотите блокировать команды `\write`, то используйте следующий прием, который придумал Frank Yellin:

```
\let\immediate=\relax \def\write#1#{{\afterassignment}\toks0=}
```

Эти изменения обычно приводят к тому, что TEX работает в четыре раза быстрее.

Вольф, обойдя стол и сев на стул,
положил на собеседника руку: "Прошу Вас,
мистер Хомберт! Я думаю, всегда лучше
перейти сразу к сути, если это возможно."

Wolfe, who had moved around the desk
and into his chair, put up a palm at him:
"Please, Mr. Hombert. I think it is
always advisable to take a short-cut
when it is feasible."

— REX STOUT, *The Rubber Band* (1936)

"Мой дорогой Уатсон, попробуйте
немного поанализировать сами," сказал он
с легким раздражением. "Вы знаете мой метод.
Примените его, и будет поучительно"
сравнить результаты."

"My dear Watson, try a little analysis yourself,"
said he, with a touch of impatience.
"You know my methods. Apply them,
and it will be instructive
to compare results."

— CONAN DOYLE, *The Sign of the Four* (1890)

E

Примеры форматов

Хотя формат начального TEX'а приложения B ориентирован на технические отчеты, его легко адаптировать и к другим задачам. Здесь представлены примеры трех форматов: (1) формат для деловых писем; (2) формат для концертной программы; (3) формат, использованный при наборе этой книги.

Сначала давайте рассмотрим деловые письма. Предположим, вам надо, чтобы TEX оформлял вашу корреспонденцию и что надо послать n писем. Это легко сделать, если система содержит файл `letterformat.tex` типа того, что описан в этом приложении позже. Надо применить TEX к файлу, который выглядит так:

⟨возможное увеличение⟩
`\input letterformat`
⟨деловое письмо$_1$⟩
⋮
⟨деловое письмо$_n$⟩
`\end`

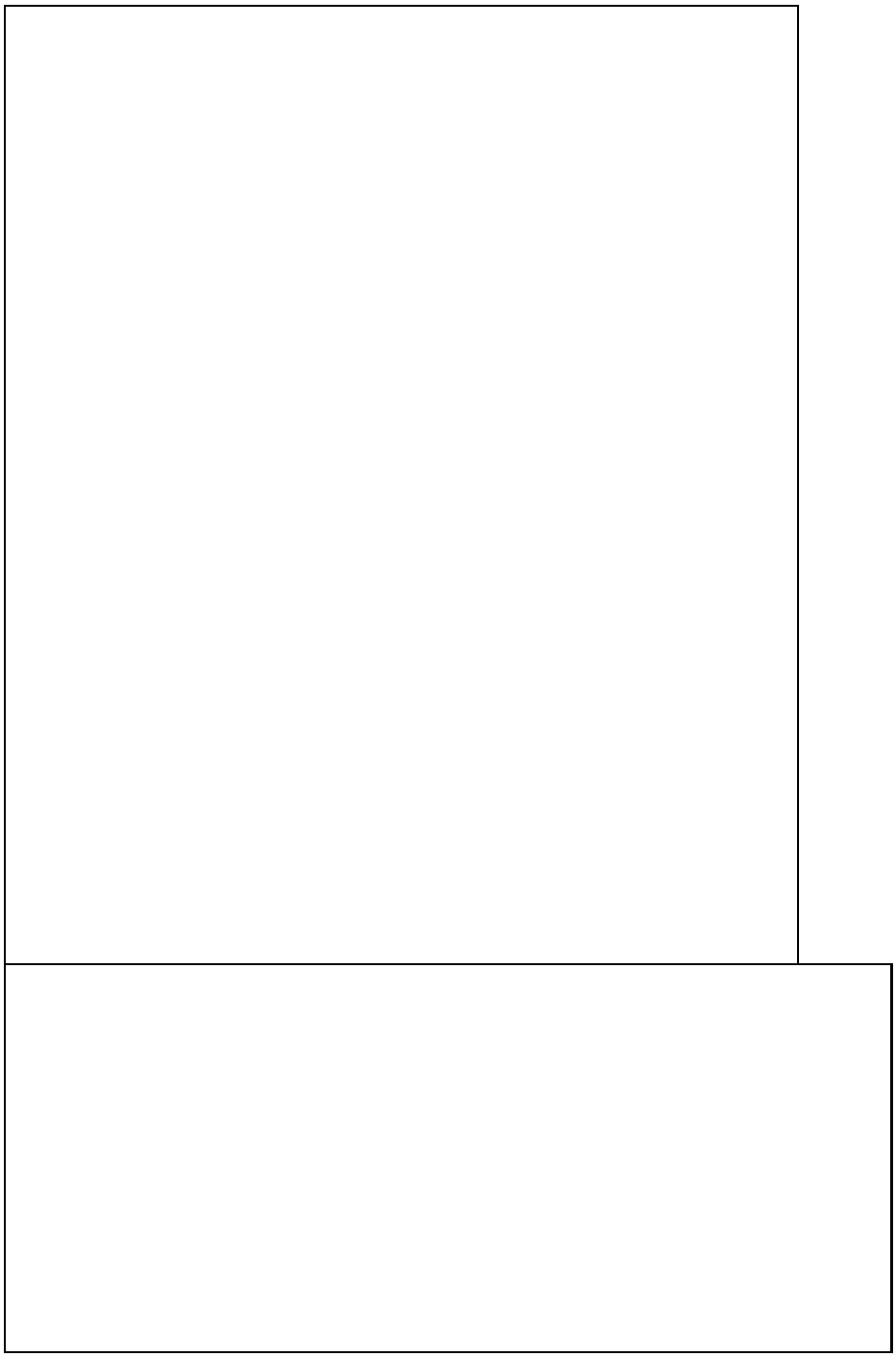
Каждое из этих n деловых писем имеет вид:

⟨заголовок письма⟩
`\address`
⟨одна или более строк адреса⟩
`\body`
⟨один или более абзацев текста⟩
`\closing`
⟨одна или более строк для приветствия и подписи⟩
⟨возможные примечания⟩
⟨возможные постскриптумы⟩
`\endletter`
`\makelabel % опустите это, если этикетка адреса не нужна`

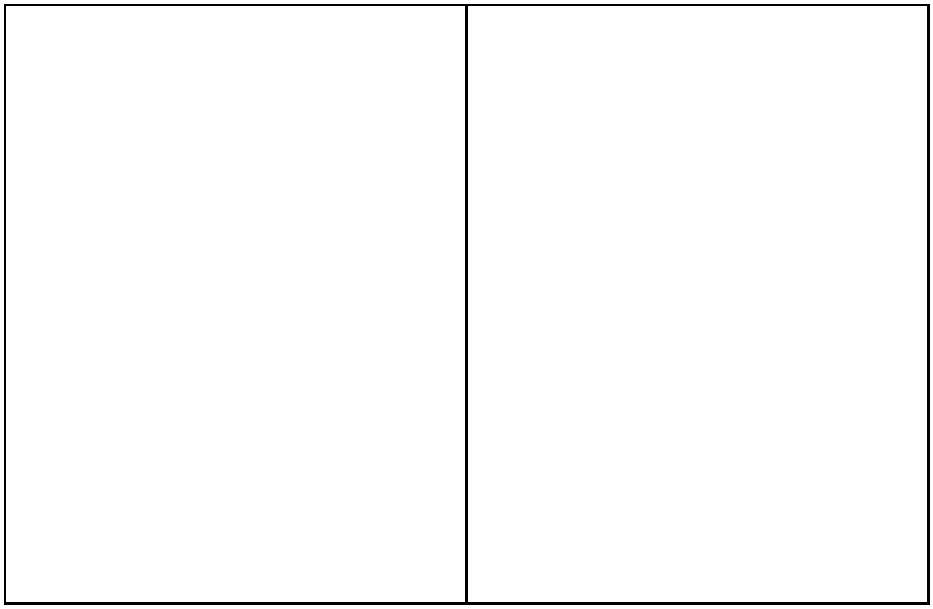
⟨заголовок письма⟩ в начале — это команда типа `\rjdletterhead` для писем автора R. J. D. Каждый автор имеет свой заголовок, который хранится в командах `letterformat`. ⟨Возможные примечания⟩ — это однострочные заметки, перед которыми стоит `\annotations`; ⟨возможные постскриптумы⟩ — это абзацы, перед которыми расположена `\ps`. Когда TEX обрабатывает `\address`, `\closing` и возможные `\annotations`, он производит строку за строкой точно так, эти как строки появляются во входном файле. Но обрабатывая `\body` и возможное `\ps`, он выравнивает строки, как когда набирает текст в книгах.

На следующих двух страницах приводится полный пример вместе с результатом. Пример начинается с `\magnification=\magstep1`, поскольку письмо довольно короткое. В длинных письмах увеличение обычно опускается. Для писем среднего размера подходит `\magnification=\magstephalf`. Но ко всем n письмам применяется одно и то же увеличение, так что если вам нужны разные увеличения, надо пропустить TEX несколько раз.

```
\magnification=\magstep1
\input letterformat

\rjdletterhead % (см. вывод на следующей странице)

\address
Проф.~Бриан~К. Рейд
Отдел электрического машиностроения
Стэнфордский Университет
Стэнфорд, CA 94305

\body
Дорогой проф.~Рейд:

Я слышал о ваших  трудностях с таблетками
Alka-Seltzer. Поскольку в каждой бутылочке
по 25~таблеток, в то время как
инструкции рекомендуют ``буль,~буль, шип,~шип'',
мои коллеги сказали мне, что у вас накопилось
значительное количество бутылочек,
в каждой из которых осталось по одной
таблетке. % (См. the 1978 SCRIBE User Manual, стр. 90.)

В настоящее время я занимаюсь исследованиями о возможных
применениях изолированных аналгетиков. Если бы вы  были так
добры, чтобы пожертвовать вашу коллекцию Alka-Seltzer
для нашего проекта, я был бы более чем счастлив посылать
вам препринты всех новейших отчетов, которые мы будем
публиковать по  этой крайне важной проблеме.

\closing
Искренне ваш,
R. J. Drofnats
Профессор

\annotations
RJD/dek
cc: {\sl The \TeX book}

\ps
P. S. \ Если  хотите, я попробую сделать так,
чтобы ваше пожертвование вместе с закуской
не облагалось налогом в связи с
нашими исследованиями.
\endletter
\makelabel
```



Если в письме более одной страницы, то адрес, дата и номер страницы появляются вверху последующих страниц. Например, если к предыдущему письму добавить дополнительные абзацы, то оно получится таким:



Макропакет `letterformat.tex`, который дает этот формат, начинается с макрокоманды, раскрывающей текущую дату.

```
\def\today{\ifcase\month\or
  Январь\or Февраль\or Март\or Апрель\or Май\or Июнь\or
  Июль\or Август\or Сентябрь\or Октябрь\or Ноябрь\or Декабрь\fi
  \space\number\day, \number\year}
```

Затем задается разметка страницы, “рваной” внизу. Используется довольно большой `\interlinepenalty`, чтобы разрыв страницы происходил между абзацами.

```
\raggedbottom
\interlinepenalty=1000
\hsize=6.25truein
\voffset=24pt
\advance\vsize by-\voffset
\parindent=0pt
\parskip=0pt
\nopagenumbers
\headline={\ifnum\pageno>1
  \tenrm В адрес \addressee\hfil\today\hfil страница \folio
  \else\hfil\fi}
```

Содержание письма печатается либо в "строковой моде" (подчиняясь строкам), либо в "абзацной моде" (производя абзацы в блочном стиле). Команды `\beginlinemode` и `\beginparmode` вводят эти моды. Команда `\endmode` определяется и переопределяется так, что текущая мода завершается правильно:

```
\def\beginlinemode{\endmode
  \begingroup\obeylines\def\endmode{\par\endgroup}}
\def\beginparmode{\endmode
  \begingroup\parskip=\medskipamount \def\endmode{\par\endgroup}}
\let\endmode=\par
\def\endletter{\endmode\vfill\supereject}
```

Одной из главных характеристик формата этого письма является параметр `\longindentation`, создающий отступ в заключающем материале и в некоторых деталях заголовка. Макрокоманда `\address` создает бокс, который используется как в письме, так и на конверте. Если строки адреса превышают `\longindentation`, они разрываются, а в переносимой части используется подвешенный отступ.

```
\newdimen\longindentation \longindentation=4truein
\newbox\theaddress
\def\address{\beginlinemode\getaddress}
{\obeylines\gdef\getaddress #1
  #2
  {#1\gdef\addressee{#2}%
    \global\setbox\theaddress=\vbox\bgroup\raggedright%
    \hsize=\longindentation \everypar{\hangindent2em}#2
    \def\endmode{\egroup\endgroup \copy\theaddress \bigskip}}}
```

(Параметр `#2` в `\getaddress` — это строка, которая следует за `\address`, т.е., адрес.)

Завершающие макрокоманды внимательно следят, чтобы не было разрыва страницы где-нибудь между концом `\body` и началом `\ps`.

```
\def\body{\beginparmode}
\def\closing{\beginlinemode\getclosing}
{\obeylines\gdef\getclosing #1
  #2
  {#1\nobreak\bigskip \leftskip=\longindentation #2
    \nobreak\bigskip\bigskip\bigskip % space for signature
    \def
    {\endgraf\nobreak}}}
\def\annotations{\beginlinemode\def\par{\endgraf\nobreak}\obeylines\par}
\def\ps{\beginparmode\nobreak
  \interlinepenalty5000\def\par{\endgraf\penalty5000}}
```

Оставшаяся часть `letterformat.tex` относится к заголовку и к этикеткам. В примере использованы макрокоманды, приведенные ниже. Их можно модифицировать более или менее очевидными способами, чтобы получать заголовки писем других видов. Обычно требуются специальные шрифты, которые должны быть загружены в `true`-размерах, чтобы не подвергаться увеличению. Здесь

стоит упомянуть еще одно маленькое усовершенствование — макрокоманду \up, которая приподнимает квадратные скобки, чтобы они лучше выглядели в номере телефона.

```
\def\up#1{\leavevmode \raise.16ex\hbox{#1}}
\font\smallheadfont=cmr8 at 8truept
\font\largeheadfont=cmdunh10 at 14.4truept
\font\logofont=manfnt at 14.4truept

\def\rjdletterhead{
  \def\sendingaddress{R. J. DROFNATS, F.T.U.G.\par
    PROFESSOR OF FARM ECOLOGY\par
    TEX.RJD @ SU-SCORE.ARPA\par
    \up[415\up]\thinspace 497-4975\par}
  \def\returnaddress{R. J. Drofnats, Dept.~of Farm Ecology\par
    The University of St.~Anford\par
    P. O. Box 1009, Haga Alto, CA 94321 USA}
  \letterhead}

\def\letterhead{\pageno=1 \def\addressee{} \univletterhead
  {\leftskip=\longindentation
    {\baselineskip9truept\smallheadfont\sendingaddress}
    \bigskip\bigskip\rm\today\bigskip}}

\def\univletterhead{\vglue-\voffset
  \hbox{\hbox to\longindentation{\raise4truemm\hbox{\logofont
        \kern2truept X\kern-1.667truept
        \lower2truept\hbox{X}\kern-1.667truept X}\hfil
      \largeheadfont The University of St.~Anford\hfil}%
    \kern-\longindentation
    \vbox{\smallheadfont\baselineskip9truept
      \leftskip=\longindentation BOX 1009\par HAGA ALTO, CA 94321}}
  \vskip2truept\hrule\vskip4truept }

\def\makelabel{\endletter\hbox{\vrule
    \vbox{\hrule \kern6truept
      \hbox{\kern6truept\vbox to 2truein{\hsize=\longindentation
          \smallheadfont\baselineskip9truept\returnaddress
          \vfill\moveright 2truein\copy\theaddress\vfill}%
        \kern6truept}\kern6truept\hrule}\vrule}
  \pageno=0\vfill\eject}
```

Второй пример — программа, которая используется на концертах оркестра, сольных концертах и так далее. Мы предполагаем, что она целиком помещается на одной странице, ширина которой 4 дюйма. Обычно используется сравнительно крупный шрифт (12 pt), но имеются и шрифты 10 pt и даже 8 pt на случай, если программа включает разделы с многими подчастями (например, Bach's Mass in B minor, или Beethoven's Diabelli Variations). Чтобы выбрать размер шрифта, пользователь говорит, соответственно \bigtype, \medtype или \smalltype. Эти макрокоманды довольно просты, поскольку программы концертов не включают математических формул, поэтому не надо менять математические шрифты. С

другой стороны, формат берет знаки диеза и бемоля из "математического курсива", который он называет `\mus`:

```
\font\twelverm=cmr12
\font\twelvebf=cmbx12
\font\twelveit=cmti12
\font\twelvesl=cmsl12
\font\twelvemus=cmmi12
\font\eightrm=cmr8
\font\eightbf=cmbx8
\font\eightit=cmti8
\font\eightsl=cmsl8
\font\eightmus=cmmi8
\def\bigtype{\let\rm=\twelverm \let\bf=\twelvebf
  \let\it=\twelveit \let\sl=\twelvesl \let\mus=\twelvemus
  \baselineskip=14pt minus 1pt
  \rm}
\def\medtype{\let\rm=\tenrm \let\bf=\tenbf
  \let\it=\tenit \let\sl=\tensl \let\mus=\teni
  \baselineskip=12pt minus 1pt
  \rm}
\def\smalltype{\let\rm=\eightrm \let\bf=\eightbf
  \let\it=\eightit \let\sl=\eightsl \let\mus=\eightmus
  \baselineskip=9.5pt minus .75pt
  \rm}
\hsize=4in
\nopagenumbers
\bigtype
```

Отметим сжимаемость в `\baselineskip`. Она не нужна в книгах, поскольку различное расстояние между строками на страницах выглядит плохо, но в одностраничном документе это помогает сжать результат, чтобы поместить его на странице. (Здесь не нужна растяжимость между базовыми линиями, поскольку внизу страницы использован `\vfill`.)

Музыкальные программы используют специальный словарь, и желательно определить несколько команд для тех вещей, которые начальный TEX делает не так удобно, как надо бы в этом конкретном приложении:

```
\def\(#1){{\rm(}#1\/{\rm)}}
\def\sharp{\raise.4ex\hbox{\mus\char"5D}}
\def\flat{\raise.2ex\hbox{\mus\char"5B}}
\let\,=\thinspace
```

Команда `\(` дает романские круглые скобки в курсивном тексте, а `\sharp` и `\flat` — музыкальные знаки в текущем шрифтовом размере. Команда `\,` задает тонкий пробел, используемый в "K. 550" и "Op. 59". (Начальный TEX определял другие `\,`, `\sharp` и `\flat`, но те определения применяются только для формул и не относятся к этому приложению.)

Перед обсуждением остальных макрокоманд, давайте посмотрим на вход и выход для типичной концертной программы:

```
\input concert

\tsaologo
\medskip
\centerline{Friday, November 19, 1982, 8:00 p.m.}
\bigskip
\centerline{\bf PROGRAM}
\medskip

\composition{Variations on a Theme by Tchaikovsky}
\composer{Anton S. Arensky (1861--1906)}
\smallskip
{\medtype
\movements{Tema: Moderato\cr
  Var.~I: Un poco pi\`u mosso&Var.~V: Andante\cr
  Var.~II: Allegro non troppo&Var.~VI: Allegro con spirito\cr
  Var.~III: Andantino tranquillo&Var.~VII: Andante con moto\cr
  Var.~IV: Vivace&Coda: Moderato\cr}
}

\bigskip

\composition{Concerto for Horn and Hardart, S.\,27}
\composer{P. D. Q. Bach (1807--1742)?}
\smallskip
\movements{Allegro con brillo\cr
  Tema con variazione \(su una tema differente)\cr
  Menuetto con panna e zucchero\cr}
\medskip
\soloists{Ben Lee User, horn\cr
  Peter Schickele, hardart\cr}

\bigskip
\centerline{INTERMISSION}
\bigskip

\composition{Symphony No.\,3 in E\flat\ Major\cr
  Op.\,55, ``The Eroica''\cr}
\composer{Ludwig van Beethoven (1770--1827)}
\smallskip
\movements{Allegro con brio\cr
  Marcia funebre: Adagio assai\cr
  Scherzo: Allegro vivace\cr
  Finale: Allegro molto\cr}

\bigskip
\smalltype \noindent
Members of the audience are kindly requested to turn off the

\bye
```

THE ST. ANFORD ORCHESTRA
R. J. Drofnats, Conductor

Friday, November 19, 1982, 8:00 p.m.

PROGRAM

Variations on a Theme by Tchaikovsky
Anton S. Arensky (1861–1906)

Tema: Moderato
Var. I: Un poco più mosso
Var. II: Allegro non troppo
Var. III: Andantino tranquillo
Var. IV: Vivace
Var. V: Andante
Var. VI: Allegro con spirito
Var. VII: Andante con moto
Coda: Moderato

Concerto for Horn and Hardart, S. 27
P. D. Q. Bach (1807–1742)?

Allegro con brillo
Tema con variazione (*su una tema differente*)
Menuetto con panna e zucchero

Ben Lee User, horn
Peter Schickele, hardart

INTERMISSION

Symphony No. 3 in E♭ Major
Op. 55, "The Eroica"
Ludwig van Beethoven (1770–1827)

Allegro con brio
Marcia funebre: Adagio assai
Scherzo: Allegro vivace
Finale: Allegro molto

Members of the audience are kindly requested to turn off the alarms on their digital watches, and to cough only between movements.

Большинство макрокоманд в `concert.tex` уже определено. Начальный TEX заботится о таких вещах как `\centerline` и `\bigskip`, так что осталось задать только `\composition`, `\composer`, `\movements` и `\soloists`:

```
\def\composition#1{\halign{\bf\quad##\hfil\cr
        \kern-1em#1\crcr}} % use \cr's if more than one line
\def\composer#1{\rightline{\bf#1}}
\def\movements#1{\halign{\quad\it##\hfil&&\qquad\it##\hfil\cr#1\crcr}}
\def\soloists#1{\centerline{\bf\vbox{\halign{##\hfil\cr#1\crcr}}}}
```

Макрокоманда `\composition` говорит помещать название произведения, если это необходимо, на двух или более строках, но обычно для этого достаточно одной строки. Заметим что `\crcr` использовано так, что конечное `\cr` в аргументе `\composition` не нужно. Аналогично, `\movements` может быть использовано, чтобы получить одну строку, а `\soloists` — когда есть только один солист.

Существует также макрокоманда `\tsaologo`. Она применяется только к одному конкретному оркестру, но определение от этого не менее интересно:

```
\def\tsaologo{\vbox{\bigtype\bf
        \line{\hrulefill}
        \kern-.5\baselineskip
        \line{\hrulefill\phantom{ THE ST.\,ANFORD ORCHESTRA }\hrulefill}
        \kern-.5\baselineskip
        \line{\hrulefill\hbox{ THE ST.\,ANFORD ORCHESTRA }\hrulefill}
        \kern-.5\baselineskip
        \line{\hrulefill\phantom{ R. J. Drofnats, Conductor }\hrulefill}
        \kern-.5\baselineskip
        \line{\hrulefill\hbox{ R. J. Drofnats, Conductor }\hrulefill}
        }}
```

Автор расширил эти макрокоманды до более усовершенствованного формата, которые включают специальные особенности для списка участников оркестра, примечаний к программе и так далее. Таким образом, очень легко напечатать маленькие буклеты для постоянных посетителей концертов. Такие расширения не стоит обсуждать в этом приложении, поскольку они не иллюстрируют никаких существенно новых идей.

Заметим, что макрокоманды `\composition`, `\movements` и `\soloists` не содержат какого-либо специального обеспечения для вертикальных пропусков. Предполагается, что пользователь будет по своему желанию вставлять `\smallskip`, `\medskip` и `\bigskip`. Это было сделано преднамеренно, поскольку различные концертные программы требуют различных пропусков. На практике ни одна автоматическая схема не работает достаточно хорошо, поскольку музыкальная литература очень разнообразна.

Давайте теперь обратимся к созданию формата для целой книги, используя к качестве примера саму эту книгу. Как автор подготовил компьютерный файл, который генерирует *TEXbook*? Мы уже видели несколько сот страниц, полученных из этого файла. В оставшейся части этого приложения мы хотим просмотреть входной файл, который был использован за кулисами.

Сначала автор подготовил страницы образцов и показал их художнику по изданию книги. (Важность этого шага нельзя недооценивать. Опасно, если автор, который теперь с помощью TEX'а может сам печатать свои книги, попытается создать свой собственный дизайн без профессиональной помощи. Книжный дизайн — это искусство, которое требует значительного творчества, мастерства, опыта и вкуса. Это одна из наиболее важных услуг, которые издатель традиционно оказывает автору.)

Образцовые страницы, которые используются как основа для дизайна, должны включать все элементы книги. В данном случае элементы включают в себя заголовки глав, иллюстрации, заголовки подглав, сноски, выделенные формулы, шрифт пишущей машинки, знаки опасного поворота, упражнения, ответы, цитаты, таблицы, перенумерованные списки, списки, помеченные точками и так далее. Автор также выразил желание иметь увеличенные поля, чтобы читатель мог делать на них заметки.

Дизайнер, Herb Caswell, столкнулся с трудной задачей по внесению всех этих несовместимых элементов в последовательную структуру. Он решил достигнуть этого, используя единообразный отступ в 3 pc как для обычного абзаца, так и для знаков опасного поворота, а также использовать этот элемент дизайна для всего выделенного материала вместо того, чтобы центрировать его.

Он решил печатать номера страниц жирным шрифтом на полях (где, благодаря требованию автора, есть достаточно места), а также использовать курсивный шрифт с заглавными и строчными буквами для бегущей верхней строки, чтобы страницы имели некоторый неформальный оттенок.

Он выбрал 10-пунктовый шрифт (на базе 12-пунктового) для основного текста и 9-пунктовый (на базе 11-пунктового) для абзацев с опасным поворотом. Шрифт был предварительно определен. Он задал `\hsize` 29 pc и `\vsize` — 44 pc. Он решил давать подзаголовки типа ▶**УПРАЖНЕНИЕ 13.8** жирными заглавными буквами перед текстом каждого упражнения. Он задал величину вертикального пробела перед и после таких элементов, как упражнения, абзацы с опасным поворотом и выделенные уравнения. Он решил посвятить левую страницу иллюстрации для каждой главы. И так далее. Каждое решение влияло на другие, чтобы окончательная книга получилась настолько последовательной и привлекательной, насколько это возможно при данных обстоятельствах. После того, как была сконструирована основная часть книги, он разработал формат для фронтального материала (то есть страниц перед первой страницей): он решил там иметь одну и ту же величину "погружения" (чистого места) вверху каждой страницы, так чтобы открывающие страницы книги выглядели унифицированными и "открытыми".

Автор в действительности не следовал во всех деталях указаниям дизайнера. Например, хотя в дизайне ничего не упоминалось о растяжимости и сжимаемости вертикальных пробелов, автор по своей инициативе ввел понятие о гибком клее, основываясь на своих наблюдениях операций разрезаний и склеивания, которые часто используются при наборе страниц. Красота этой книги должна быть приписана Herb Caswell'y, а если она имеет какие-нибудь недостатки, то это вина Дона Кнута, написавшего форматирующие макрокоманды, которые мы сейчас будем обсуждать.

Компьютерный файл `manual.tex`, который генерировал *TEXbook*, начинает с уведомления об авторском праве, а затем говорит `\input manmac`. Вспомога-

тельный файл `manmac.tex` содержит форматирующие макрокоманды и начинается загрузкой 9-пунктовых, 8-пунктовых и 6-пунктовых шрифтов:

```
\font\ninerm=cmr9   \font\eightrm=cmr8   \font\sixrm=cmr6
\font\ninei=cmmi9   \font\eighti=cmmi8   \font\sixi=cmmi6
\font\ninesy=cmsy9  \font\eightsy=cmsy8  \font\sixsy=cmsy6
\font\ninebf=cmbx9  \font\eightbf=cmbx8  \font\sixbf=cmbx6
\font\ninett=cmtt9  \font\eighttt=cmtt8
\font\nineit=cmti9  \font\eightit=cmti8
\font\ninesl=cmsl9  \font\eightsl=cmsl8
```

(Эти шрифты были `\preloaded` в приложении B, а теперь они загружены официально.)

Шрифтам, предназначенным для математических формул, нужно иметь нестандартный `\skewchar`. Шрифтам пишущей машинки задано `\hyphenchar=-1`, так что когда имена команд и ключевые слова встречаются в тексте абзаца, их перенос запрещен.

```
\skewchar\ninei='177  \skewchar\eighti='177  \skewchar\sixi='177
\skewchar\ninesy='60  \skewchar\eightsy='60  \skewchar\sixsy='60
\hyphenchar\ninett=-1 \hyphenchar\eighttt=-1 \hyphenchar\tentt=-1
```

Для специальных целей нужно еще несколько шрифтов:

```
\font\tentex=cmtex10               % TeX character set as in Appendix C
\font\inchhigh=cminch              % inch-high caps for chapter openings
\font\titlefont=cmssdc10 at 40pt   % titles in chapter openings
\font\eightss=cmssq8               % quotations in chapter closings
\font\eightssi=cmssqi8             % ditto, slanted
\font\tenu=cmu10                   % unslanted text italic
\font\manual=manfnt                % METAFONT logo and special symbols
\font\magnifiedfiverm=cmr5 at 10pt % to demonstrate magnification
```

Теперь мы подошли к макрокомандам, переключающим размеры, которые намного сложнее команд из предыдущего примера, поскольку математику необходимо поддерживать в трех различных размерах. Формат предусматривает также псевдо “маленькие заглавные буквы” (`\sc`). Истинный заглавный-и-маленький-заглавный шрифт реально не является необходимым в тех немногих случаях, когда использовано `\sc`. Переменная размера `\ttglue` устанавливается равной желательным пробелам для текста, набранного шрифтом пишущей машинки, который время от времени появляется в абзацах. Шрифты `\tt` имеют фиксированные пробелы, которые плохо комбинируются с переменными пробелами, следовательно, макрокоманды, приведенные ниже, используют между словами в подходящих местах `\ttglue`.

```
\catcode`@=11 % we will access private macros of plain TeX (carefully)
\newskip\ttglue

\def\tenpoint{\def\rm{\fam0\tenrm}% switch to 10-point type
  \textfont0=\tenrm \scriptfont0=\sevenrm \scriptscriptfont0=\fiverm
  \textfont1=\teni  \scriptfont1=\seveni  \scriptscriptfont1=\fivei
  \textfont2=\tensy \scriptfont2=\sevensy \scriptscriptfont2=\fivesy
```

```
  \textfont3=\tenex \scriptfont3=\tenex   \scriptscriptfont3=\tenex
  \textfont\itfam=\tenit  \def\it{\fam\itfam\tenit}%
  \textfont\slfam=\tensl  \def\sl{\fam\slfam\tensl}%
  \textfont\ttfam=\tentt  \def\tt{\fam\ttfam\tentt}%
  \textfont\bffam=\tenbf  \scriptfont\bffam=\sevenbf
   \scriptscriptfont\bffam=\fivebf  \def\bf{\fam\bffam\tenbf}%
  \tt \ttglue=.5em plus.25em minus.15em
  \normalbaselineskip=12pt
  \setbox\strutbox=\hbox{\vrule height8.5pt depth3.5pt width0pt}%
  \let\sc=\eightrm  \let\big=\tenbig  \normalbaselines\rm}
\def\ninepoint{\def\rm{\fam0\ninerm}% switch to 9-point type
  \textfont0=\ninerm \scriptfont0=\sixrm \scriptscriptfont0=\fiverm
  \textfont1=\ninei  \scriptfont1=\sixi  \scriptscriptfont1=\fivei
  \textfont2=\ninesy \scriptfont2=\sixsy \scriptscriptfont2=\fivesy
  \textfont3=\tenex  \scriptfont3=\tenex \scriptscriptfont3=\tenex
  \textfont\itfam=\nineit  \def\it{\fam\itfam\nineit}%
  \textfont\slfam=\ninesl  \def\sl{\fam\slfam\ninesl}%
  \textfont\ttfam=\ninett  \def\tt{\fam\ttfam\ninett}%
  \textfont\bffam=\ninebf  \scriptfont\bffam=\sixbf
   \scriptscriptfont\bffam=\fivebf  \def\bf{\fam\bffam\ninebf}%
  \tt \ttglue=.5em plus.25em minus.15em
  \normalbaselineskip=11pt
  \setbox\strutbox=\hbox{\vrule height8pt depth3pt width0pt}%
  \let\sc=\sevenrm  \let\big=\ninebig  \normalbaselines\rm}
\def\eightpoint{\def\rm{\fam0\eightrm}% switch to 8-point type
  \textfont0=\eightrm \scriptfont0=\sixrm \scriptscriptfont0=\fiverm
  \textfont1=\eighti  \scriptfont1=\sixi  \scriptscriptfont1=\fivei
  \textfont2=\eightsy \scriptfont2=\sixsy \scriptscriptfont2=\fivesy
  \textfont3=\tenex   \scriptfont3=\tenex \scriptscriptfont3=\tenex
  \textfont\itfam=\eightit  \def\it{\fam\itfam\eightit}%
  \textfont\slfam=\eightsl  \def\sl{\fam\slfam\eightsl}%
  \textfont\ttfam=\eighttt  \def\tt{\fam\ttfam\eighttt}%
  \textfont\bffam=\eightbf  \scriptfont\bffam=\sixbf
   \scriptscriptfont\bffam=\fivebf  \def\bf{\fam\bffam\eightbf}%
  \tt \ttglue=.5em plus.25em minus.15em
  \normalbaselineskip=9pt
  \setbox\strutbox=\hbox{\vrule height7pt depth2pt width0pt}%
  \let\sc=\sixrm  \let\big=\eightbig  \normalbaselines\rm}
\def\tenbig#1{{\hbox{$\left#1\vbox to8.5pt{}\right.\n@space$}}}
\def\ninebig#1{{\hbox{$\textfont0=\tenrm\textfont2=\tensy
  \left#1\vbox to7.25pt{}\right.\n@space$}}}
\def\eightbig#1{{\hbox{$\textfont0=\ninerm\textfont2=\ninesy
  \left#1\vbox to6.5pt{}\right.\n@space$}}}
\def\tenmath{\tenpoint\fam-1 } % for 10-point math in 9-point territory
```

метки на полях

Выпуски макетов страниц делаются следующими макрокомандами. Сначала приведем основные:

```
\newdimen\pagewidth \newdimen\pageheight \newdimen\ruleht
\hsize=29pc  \vsize=44pc  \maxdepth=2.2pt  \parindent=3pc
\pagewidth=\hsize \pageheight=\vsize \ruleht=.5pt
\abovedisplayskip=6pt plus 3pt minus 1pt
\belowdisplayskip=6pt plus 3pt minus 1pt
\abovedisplayshortskip=0pt plus 3pt
\belowdisplayshortskip=4pt plus 3pt
```

(Любопытное значение `\maxdepth` выбрано только для того, чтобы обеспечить пример в главе 15; более глубоких причин для этого нет.)

Когда автор готовил эту книгу, он на каждой странице отмечал то, что должно быть помещено в предметный указатель. В корректуре эти метки печатаются мелким шрифтом, как слова “метки на полях” на правом поле этой страницы. Для того, чтобы обращаться с такими метками, формат `manmac` использует класс вставок, называемый `\margin`.

```
\newinsert\margin
\dimen\margin=\maxdimen % no limit on the number of marginal notes
\count\margin=0 \skip\margin=0pt % marginal inserts take up no space
```

Макрокоманда `\footnote` начального TEX'а нуждается в изменении, поскольку сноски печатаются с отступом и им задан шрифт в 8 пунктов. Сделаны также некоторые упрощения, поскольку сноски в этой книге используются не так уж часто.

```
\def\footnote#1{\edef\@sf{\spacefactor\the\spacefactor}#1\@sf
      \insert\footins\bgroup\eightpoint
      \interlinepenalty100 \let\par=\endgraf
        \leftskip=0pt \rightskip=0pt
        \splittopskip=10pt plus 1pt minus 1pt \floatingpenalty=20000
        \smallskip\item{#1}\bgroup\strut\aftergroup\@foot\let\next}
\skip\footins=12pt plus 2pt minus 4pt % space added when footnote exists
\dimen\footins=30pc % maximum footnotes per page
```

Бегущий заголовок хранится в команде `\rhead`. Некоторые страницы не должны иметь заголовков: макрокоманда `\titlepage` подавляет текущий заголовок на той странице, которая будет выводится после заглавной страницы главы.

```
\newif\iftitle
\def\titlepage{\global\titletrue} % for pages without headlines
\def\rhead{} % \rhead contains the running headline

\def\leftheadline{\hbox to \pagewidth{%
    \vbox to 10pt{}% strut to position the baseline
    \llap{\tenbf\folio\kern1pc}% folio to left of text
    \tenit\rhead\hfil}} % running head flush left
\def\rightheadline{\hbox to \pagewidth{\vbox to 10pt{}%
    \hfil\tenit\rhead\/% running head flush right
    \rlap{\kern1pc\tenbf\folio}}} % folio to right of text
```

Страницы выводятся при помощи макрокоманды `\onepageout`, присоединяющей заголовки, заметки на полях и/или соответствующие сноски. Вверху титульных страниц печатаются специальные регистрационные знаки, так что страницы будут правильно выстраиваться на печатных полосах, которые перефотографируются из готового для пересъемки вывода TEX'а. В углу печатается маленький номер страницы. Эта вспомогательная информация будет конечно же удалена перед тем, как страницы будут печататься.

```
\def\onepageout#1{\shipout\vbox{ % here we define one page of output
    \offinterlineskip % butt the boxes together
    \vbox to 3pc{ % this part goes on top of the 44pc pages
      \iftitle \global\titlefalse \setcornerrules
      \else\ifodd\pageno\rightheadline\else\leftheadline\fi\fi \vfill}
    \vbox to \pageheight{
      \ifvoid\margin\else % marginal info is present
       \rlap{\kern31pc\vbox to0pt{\kern4pt\box\margin \vss}}\fi
      #1 % now insert the main information
      \ifvoid\footins\else % footnote info is present
       \vskip\skip\footins \kern-3pt
       \hrule height\ruleht width\pagewidth \kern-\ruleht \kern3pt
       \unvbox\footins\fi
      \boxmaxdepth=\maxdepth}}
  \advancepageno}

\def\setcornerrules{\hbox to \pagewidth{% for camera alignment
    \vrule width 1pc height\ruleht \hfil \vrule width 1pc}
  \hbox to \pagewidth{\llap{\sevenrm(page \folio)\kern1pc}%
    \vrule height1pc width\ruleht depth0pt
    \hfil \vrule width\ruleht depth0pt}}

\output{\onepageout{\unvbox255}}
```

Для приложения I (предметного указателя) необходима другая программа вывода, поскольку большая часть этого приложения печатается в двухколоночном формате. Вместо того, чтобы справляться с двойными колонками при помощи переключения `\lr`, как обсуждалось в главе 23, `manmac` делает это командой `\vsplit`, когда набирается достаточно материала, чтобы заполнить страницу. Этот подход делает сравнительно легким уравнивание колонок на последней странице предметного указателя. Более трудный подход был бы необходим, если бы указатель содержал вставки (например, сноски); к счастью, этого нет. Более того, нет необходимости использовать `\mark`, как предполагалось в примере указателя в главе 23, поскольку элементы в приложении I достаточно короткие. Единственная сложность, с которой встречается `manmac` — это то, что приложение I начинается и оканчивается одноколоночным форматом. С комбинированными страницами приходится обращаться очень аккуратно, так чтобы формат менялся в одну и другую стороны.

```
\newbox\partialpage
\def\begindoublecolumms{\begingroup
  \output={\global\setbox\partialpage=\vbox{\unvbox255\bigskip}}\eject
  \output={\doublecolumnout} \hsize=14pc \vsize=89pc}
```

```
\def\enddoublecolumns{\output={\balancecolumns}\eject
  \endgroup \pagegoal=\vsize}
\def\doublecolumnout{\splittopskip=\topskip \splitmaxdepth=\maxdepth
  \dimen@=44pc \advance\dimen@ by-\ht\partialpage
  \setbox0=\vsplit255 to\dimen@ \setbox2=\vsplit255 to\dimen@
  \onepageout\pagesofar \unvbox255 \penalty\outputpenalty}
\def\pagesofar{\unvbox\partialpage
  \wd0=\hsize \wd2=\hsize \hbox to\pagewidth{\box0\hfil\box2}}
\def\balancecolumns{\setbox0=\vbox{\unvbox255} \dimen@=\ht0
  \advance\dimen@ by\topskip \advance\dimen@ by-\baselineskip
  \divide\dimen@ by2 \splittopskip=\topskip
  {\vbadness=10000 \loop \global\setbox3=\copy0
    \global\setbox1=\vsplit3 to\dimen@
    \ifdim\ht3>\dimen@ \global\advance\dimen@ by1pt \repeat}
  \setbox0=\vbox to\dimen@{\unvbox1} \setbox2=\vbox to\dimen@{\unvbox3}
  \pagesofar}
```

Балансирование устанавливает `\vbadness` бесконечной в то время, когда ищется подходящая высота колонки, так что сообщений о переполнении v-боксов не будет, если только колонки не получаются после балансирования действительно плохими. Колонки в приложении I имеют большую растяжимость, поскольку между соседними элементами имеется `\parskip` величиной `0pt plus .8pt` и в одной колонке помещается более 50 строк. Поэтому балансирующая программа `manmac` пытается сделать так, чтобы в конце указателя согласовывались как верхние, так и нижние базовые линии. В тех случаях, когда клей не такой гибкий, лучше было бы позволить правой колонке быть немного короче. Это, вероятно, лучше всего сделать, заменив команду `\unvbox3` на `\dimen2=\dp3 \unvbox3 \kern-\dimen2 \vfil`.

Следующие макрокоманды относятся к форматированию глав. Каждая глава в рукописи начинается с макрокоманды `\beginchapter` и оканчивается макрокомандой `\endchapter` и двумя цитатами, за которыми следует `\eject`. Например, главу 15 образована командами TEX'а, которые в файле `manual.tex` выглядят так*:

```
\beginchapter Глава 15. Как \TeX\ собирает\\строки в  страницы

\TeX\ пытается выбрать подходящее место, чтобы поделить ваш документ
    ...  (здесь опущено более 1100 строк рукописи)
страницы. \ (Глубокий вдох.) \ Вы усвоили это?

\endchapter

Since it is impossible to foresee how [footnotes] will happen to come out
in the make-up, it is impracticable to number them from 1 up on each page.
The best way is to number them consecutively throughout an article
or by chapters in a book.
\author UNIVERSITY OF CHICAGO PRESS, {\sl Manual of Style\/} (1910)
```

* Имеется ввиду подготовка оригинал-макета английского издания. В русском переводе некоторые макрокоманды были изменены. (*прим. переводчика*)

```
\bigskip

Don't use footnotes in your books, Don.
\author JILL KNUTH (1962)

\eject
```

В строке заголовка \\ задает разрыв строки, которая печатается на левосторонней титульной странице в начале главы. Большая часть макрокоманды \beginchapter посвящена приготовлению этой титульной страницы. Слову \TeX, когда оно печатается в \titlefont, необходимы немного необычные пробелы, а цифры \inchhigh, чтобы правильно выглядеть в заголовке, должны быть ближе друг к другу.

```
\newcount\exno % for the number of exercises in the current chapter
\newcount\subsecno % for the number of subsections in the current chapter

\outer\def\beginchapter#1 #2#3. #4\par{\def\chapno{#2#3}
  \global\exno=0 \subsecno=0
  \ifodd\pageno
   \errmessage{You had too much text on that last page; I'm backing up}
   \advance\pageno by-1 \fi
  \def\\{ } % \\'s in the title will be treated as spaces
  \message{#1 #2#3:} % show the chapter title on the terminal
  \xdef\rhead{#1 #2#3: #4\unskip} % establish a new running headline
  {\def\TeX{T\kern-.2em\lower.5ex\hbox{E}\kern-.06em X}
    \def\\{#3}
    \ifx\empty\\ \rightline{\inchhigh #2\kern-.04em}
    \else\rightline{\inchhigh #2\kern-.06em#3\kern-.04em}\fi
    \vskip1.75pc \baselineskip=36pt \lineskiplimit=1pt \lineskip=12pt
    \let\\=\cr % now the \\'s are line dividers
    \halign{\line{\titlefont\hfil##}\\#4\unskip\\}
    \titlepage\vfill\eject} % output the chapter title page
  \tenpoint\noindent\ignorespaces} % the first paragraph is not indented
```

Если необходимо, в конце главы выводится дополнительная страница, чтобы завершающие цитаты оказались на правосторонней странице. (Логика для выполнения этого не безупречна, но ей и не надо быть таковой, поскольку она не срабатывает только тогда, когда глава должна каким-нибудь образом укорачиваться или удлиняться. Подготовка книги при помощи TEX'а поощряет взаимодействие между человеком и машиной.) Цитаты прижимаются вправо с помощью \obeylines вместе с растяжимым \leftskip:

```
\outer\def\endchapter{\ifodd\pageno \else\vfill\eject\null\fi
  \begingroup\bigskip\vfill % beginning of the quotes
  \def\eject{\endgroup\eject} % ending of the quotes
  \def\par{\ifhmode\/\endgraf\fi}\obeylines
  \def\TeX{T\kern-.2em\lower.5ex\hbox{E}X}
  \eightpoint \let\tt=\ninett \baselineskip=10pt \interlinepenalty=10000
  \leftskip=0pt plus 40pc minus \parindent \parfillskip=0pt
  \let\rm=\eightss \let\sl=\eightssi \everypar{\sl}}
\def\author#1(#2){\smallskip\noindent\rm--- #1\unskip\enspace(#2)}
```

Мы подошли теперь к тому, что находится внутри самих глав. Знаки опасного поворота и двойные знаки опасного поворота задаются командами `\danger` или `\ddanger` перед абзацем, который надо выделить предупреждающим символом:

```
\def\dbend{{\manual\char127}} % "dangerous bend" sign
\def\d@nger{\medbreak\begingroup\clubpenalty=10000
  \def\par{\endgraf\endgroup\medbreak} \noindent\hang\hangafter=-2
  \hbox to0pt{\hskip-\hangindent\dbend\hfill}\ninepoint}
\outer\def\danger{\d@nger}
\def\dd@nger{\medbreak\begingroup\clubpenalty=10000
  \def\par{\endgraf\endgroup\medbreak} \noindent\hang\hangafter=-2
  \hbox to0pt{\hskip-\hangindent\dbend\kern1pt\dbend\hfill}\ninepoint}
\outer\def\ddanger{\dd@nger}
\def\enddanger{\endgraf\endgroup} % omits the \medbreak
```

(`\enddanger` в конце опасного поворота необходимо только в тех редких случаях, когда после абзаца не желателен средний пробел. Например, `\smallskip\item` мог быть использован, чтобы задать перечисление по пунктам в пределах знака опасного поворота.)

Некоторые главы и приложения в этой книге поделены на нумерованные подсекции (например, глава 18 и приложение B). Такие подсекции в рукописи задаются, например, так:

```
\subsection Назначение регистров.
```

Приложение A разделено другим способом на абзацы, которые имеют номер ответа:

```
\outer\def\subsection#1. {\medbreak\advance\subsecno by 1
  \noindent{\it \the\subsecno.\enspace#1.\enspace}}
\def\ansno#1.#2:{\medbreak\noindent
  \hbox to\parindent{\bf\hss#1.#2.\enspace}\ignorespaces}
```

Ниже мы увидим, что в рукописи в действительности не задается `\ansno` прямо. Каждый вызов `\ansno` генерируется автоматически макрокомандой `\answer`.

Приложение H указывает, что в *TEXbook* требуется три исключения для переноса слов:

```
\hyphenation{man-u-script man-u-scripts ap-pen-dix}
```

Некоторые макрокоманды в `manmac`'е призваны обеспечивать специальные конструкции, которые время от времени нужны в тексте книги: `\MF` для METAFONT, `\AmSTeX` для $\mathcal{AMS}$-TEX, `\bull` для •, `\dn` и `\up` для ↓и ↑, `\|` и `\]` для | и ␣. Чтобы напечатать

3 pt of ⟨stuff⟩, *'105* = 69, *"*69 = 105, wow

рукопись говорит

```
$3\pt$ of \<stuff>, $\oct{105}=69$, $\hex{69}=105$, \cstok{wow}
```

используя макрокоманды `\pt`, `\<`, `\oct`, `\hex` и `\cstok`.

```
\def\MF{{\manual META}\-{\manual FONT}}
```

```
\def\AmSTeX{$\cal A\kern-.1667em\lower.5ex\hbox{$\cal M$}\kern-.075em
  S$-\TeX}
\def\bull{\vrule height .9ex width .8ex depth -.1ex } % square bullet
\def\SS{{\it SS}} % scriptscript style
\def\dn{\leavevmode\hbox{\tt\char'14}} % downward arrow
\def\up{\leavevmode\hbox{\tt\char'13}} % upward arrow
\def\|{\leavevmode\hbox{\tt\char`\|}} % vertical line
\def\]{\leavevmode\hbox{\tt\char`\ }} % visible space

\def\pt{\,{\rm pt}} % units of points, in math formulas
\def\<#1>{\leavevmode\hbox{$\langle$#1\/$\rangle$}} % syntactic quantity
\def\oct#1{\hbox{\rm\'{}\kern-.2em\it#1\/\kern.05em}} % octal constant
\def\hex#1{\hbox{\rm\H{}\tt#1}} % hexadecimal constant
\def\cstok#1{\leavevmode\thinspace\hbox{\vrule\vtop{\vbox{\hrule\kern1pt
       \hbox{\vphantom{\tt/}\thinspace{\tt#1}\thinspace}}
      \kern1pt\hrule}\vrule}\thinspace} % control sequence token
```

В этой книге выделенный материал обычно чаще напечатан с отступом, чем центрирован и чаще содержит текст, чем математику. В формате `manmac` удобно делать такие выделения с помощью двух макрокоманд `\begindisplay` и `\enddisplay`. Имеется также пара макрокоманд `\begintt` и `\endtt` для выделенного материала, который целиком набран шрифтом пишущей машинки.Такие выделения копируются дословно из файла рукописи без какой-либо специальной интерпретации символов типа `\` или `$`. Например, часть верхнего абзаца была напечатана так:

```
... Чтобы напечатать
\begindisplay
$3\pt$ of \<stuff>, $\oct{105}=69$, $\hex{69}=105$, \cstok{wow}
\enddisplay
рукопись говорит
\begintt
$3\pt$ of \<stuff>, $\oct{105}=69$, $\hex{69}=105$, \cstok{wow}
\endtt
используя макрокоманды |\pt|, |\<|, |\oct|, |\hex| и |\cstok|.
```

(Последняя строка этого примера иллюстрирует тот факт, что дословный текст шрифта пишущей машинки внутри абзаца может быть получен, используя в качестве скобок вертикальные линии.) Макрокоманда `\begindisplay` в действительности является более общей, чем мы могли ожидать из этого примера. Она позволяет многострочные выделения с `\cr`, следующим за каждой строкой, а также позволяет, чтобы локальные определения (которые применяются только внутри выделения) были указаны сразу после `\begindisplay`.

```
\outer\def\begindisplay{\obeylines\startdisplay}
{\obeylines\gdef\startdisplay#1
  {\catcode`\^^M=5$$#1\halign\bgroup\indent##\hfil&&\qquad##\hfil\cr}}
\outer\def\enddisplay{\crcr\egroup$$}

\chardef\other=12
\def\ttverbatim{\begingroup \catcode`\\=\other \catcode`\{=\other
```

```
  \catcode`\}=\other \catcode`\$=\other \catcode`\&=\other
  \catcode`\#=\other \catcode`\%=\other \catcode`\~=\other
  \catcode`\_=\other \catcode`\^=\other
  \obeyspaces \obeylines \tt}
{\obeyspaces\gdef {\ }} % \obeyspaces now gives \ , not \space

\outer\def\begintt{$$\let\par=\endgraf \ttverbatim \parskip=0pt
  \catcode`\|=0 \rightskip=-5pc \ttfinish}
{\catcode`\|=0 |catcode`|\=\other % | is temporary escape character
  |obeylines % end of line is active
  |gdef|ttfinish#1^^M#2\endtt{#1|vbox{#2}|endgroup$$}}

\catcode`\|=\active
{\obeylines\gdef|{\ttverbatim\spaceskip=\ttglue\let^^M=\ \let|=\endgroup}}
```

Эти макрокоманды тоньше других в этом приложении и заслуживают внимательного изучения, поскольку они показывают, как исключить нормальное форматирование TEX'а. Символ | в формате `manmac` обычно является активным (категория 13) и указывает макрокоманде `\ttverbatim` преобразовать все другие необычные символы в нормальные символы (категория 12). Однако, в границах `\begintt...\endtt` вертикальная черта является сигнальным символом (категория 0), что позволяет избавиться от дословной моды.

Макрокоманда `\begintt` предполагает, что будет выделяться сравнительно маленький объем текста. Строки дословно помещаются в v-бокс, так что они не могут быть разбиты между страницами. В основном для текста, напечатанного в этом приложении и приложении B шрифтом пишущей машинки, использован другой подход: материал, который цитируется из форматного файла, ограничивается `\beginlines` и `\endlines`, между которыми можно давать команды типа `\smallbreak`, чтобы помочь распределению пробелов и разбиению страниц. К тому же макрокоманды `\beginlines` и `\endlines` до и после выделения печатают горизонтальную черту:

```
\def\beginlines{\par\begingroup\nobreak\medskip\parindent=0pt
  \hrule\kern1pt\nobreak \obeylines \everypar{\strut}}
\def\endlines{\kern1pt\hrule\endgroup\medbreak\noindent}
```

Например, предыдущие три строки были напечатаны следующей спецификацией

```
\beginlines
|\def\beginlines{\par\begingroup\nobreak\medskip\parindent=0pt|
\nobreak
|  \hrule\kern1pt\nobreak \obeylines \everypar{\strut}}|
|\def\endlines{\kern1pt\hrule\endgroup\medbreak\noindent}|
\endlines
```

На каждой строке помещена подпорка, поэтому горизонтальные черты помещены в правильное положение. В формат `manmac` также включает в себя макрокоманды `\beginmathdemo...\endmathdemo`, с помощью которых получались формулы в главах 16–19, `\beginsyntax...\endsyntax` для формального синтаксиса в главах 24–26, `\beginchart...\endchart` для таблиц шрифта в приложениях C и F, и так далее. Эти макрокоманды сравнительно просты и их здесь можно не показывать.

Для печати упражнений используется макрокоманда `\exercise`. Например, первое упражнение главы 1 было генерировано следующими строками рукописи:

```
\exercise После того, как вы освоите материал этой книги,
кто вы будете:  \TeX перт или  \TeX ник?
\answer  \TeX ник (низкооплачиваемый), иногда также называемый
 \TeX нарь.
```

Заметим, что `\answer` дается сразу после каждого упражнения, что облегчает вставку новых упражнений или удаление старых без изменения номеров упражнений. Упражнения, помеченные знаками опасного поворота или двойного опасного поворота вводятся макрокомандами `\dangerexercise` и `\ddangerexercise`.

```
\outer\def\exercise{\medbreak \global\advance\exno by 1
  \noindent\llap{\manual\char'170\rm\kern.15em}% triangle in margin
  {\ninebf EXERCISE \bf\chapno.\the\exno}\par\nobreak\noindent}
\def\dexercise{\global\advance\exno by 1
  \llap{\manual\char'170\rm\kern.15em}% triangle in indented space
  {\eightbf EXERCISE \bf\chapno.\the\exno}\hfil\break}
\outer\def\dangerexercise{\d@nger \dexercise}
\outer\def\ddangerexercise{\dd@nger \dexercise}
```

(Последние две строки используют `\d@nger` и `\dd@nger`, которые являются не-`\outer` эквивалентами `\danger` и `\ddanger`. Такое дублирование необходимо, поскольку команды типа `\outer` не могут появляться внутри `\def`.)

Макрокоманда `\answer` пишет ответ в файл `answers.tex`, затем приложение A читает этот файл командами `\immediate\closeout\ans \ninepoint \input answers`. Каждый ответ оканчивается чистой строкой. Таким образом, в длинном ответе между абзацами должен быть использован `\par`.

```
\newwrite\ans
\immediate\openout\ans=answers % file for answers to exercises
\outer\def\answer{\par\medbreak
  \immediate\write\ans{}
  \immediate\write\ans{\string\ansno\chapno.\the\exno:}
  \copytoblankline}
\def\copytoblankline{\begingroup\setupcopy\copyans}
\def\setupcopy{\def\do##1{\catcode`##1=\other}\dospecials
  \catcode`\|=\other \obeylines}
{\obeylines \gdef\copyans#1
  {\def\next{#1}%
  \ifx\next\empty\let\next=\endgroup %
  \else\immediate\write\ans{\next} \let\next=\copyans\fi\next}}
```

Заметим, что здесь для дословного копирования использовано `\dospecials`. Макрокоманда `\ttverbatim` также могла вызвать `\dospecials`, но `\ttverbatim` используется слишком часто, поэтому для быстродействия было введено такое упрощение.

Оставшиеся макрокоманды в формате `manmac` созданы, чтобы помочь сделать хороший предметный указатель. Когда абзац содержит слово или группу

слов, которая используется в предметном указателе, в рукописи это вставляется в `^{...}`. Например, в конце первого предложения этого абзаца в действительности стоит "`хороший предметный ^{указатель}`". Это указывает на то, что соответствующий элемент записан в файл `index.tex`, когда TEX набирал страницу. При этом также помещается слово "указатель" на полях корректуры, так что автор может вспомнить, что было отмечено для указателя, не заглядывая в файл рукописи. Указание при помощи `^{...}` не изменяет существенно поведение TEX'а. Слово "указатель" появляется как в тексте, так и в указателе.

```
\newwrite\inx
\immediate\openout\inx=index % file for index reminders
\def\marginstyle{\sevenrm % marginal notes are in 7-point type
  \vrule height6pt depth2pt width0pt } % a strut for \insert\margin
```

Иногда желательно указать слова, которые реально на странице не появляются. Примечание `^^{...}` обозначает "молчаливый" элемент указателя. В этом случае после завершающего "`}`" пробелы игнорируются. Например, в приложении I указывается страница 1 около слова "красота", даже хотя страница 1 содержит только слово "прекрасное". В рукописи это достигается так: "`прекрасное ^^{красота}`'. (Автор чувствовал, что важно поместить в указатель слово "красота", поскольку он уже поместил туда слово "истина".)

Не трудно превратить `^` в активный символ, который производит такие элементы указателя, хотя остается и использование его для верхнего индекса в математических формулах, поскольку можно использовать `\ifmmode`, чтобы проверить, используется ли команда в математической моде. Однако то, что `manmac` использует `^` как активный символ, означает, что `^^M` не может быть использовано для ссылки на символ ⟨return⟩. К счастью, запись `^^M` не является необходимой, за тем исключением, когда определяются сами форматирующие макрокоманды.

Следующие макрокоманды устанавливают, как `^` и `^^` вне математической моды соответственно преобразуются в `\silentfalse\xref` и `\silenttrue\xref`:

```
\newif\ifsilent
\def\specialhat{\ifmmode\def\next{^}\else\let\next=\beginxref\fi\next}
\def\beginxref{\futurelet\next\beginxrefswitch}
\def\beginxrefswitch{\ifx\next\specialhat\let\next=\silentxref
  \else\silentfalse\let\next=\xref\fi \next}
\catcode`\^=\active \let ^=\specialhat
\def\silentxref^{\silenttrue\xref}
```

Элементами указателя не всегда являются слова в романском шрифте. Могут потребоваться специальные соглашения. Например, в приложении I есть множество элементов, которые начинаются с обратной косой черты и напечатаны шрифтом пишущей машинки. Формат `manmac` легко получает такие элементы, печатая, например `^|\immediate|` вместо `^{|\immediate|}`. В этом случае обратная косая черта не записывается в файл `index`, поскольку это перепутало бы алфавитный порядок элементов. Номер кода переписывается, так что обратная косая черта может быть восстановлена после того, как указатель отсортирован. Номер кода также используется, чтобы напечатать элемент шрифтом пишущей машинки.

Макрокоманды предметного указателя в `manmac`'е производят элементы четырех видов, которым присвоены коды 0, 1, 2 и 3. Код 0 применяется, когда

аргумент заключается в фигурные скобки, например, `^{слово}`. Код 1 применяется, когда аргумент заключается в вертикальные черточки и не имеет обратной косой черты, например, `^|plus|`. Код 2 аналогичен, но с обратной косой чертой, например, `^|\par|`. Код 3 применяется, когда аргумент заключается в угловые скобки, например, `^\<stuff>`. Четыре примера элементов в предыдущем предложении будут записаны в файл `index.tex` в виде

```
слово !0 123.
plus !1 123.
par !2 123.
stuff !3 123.
```

если они появляются на странице 123 книги.

```
\chardef\bslash=`\\ % \bslash makes a backslash (in tt fonts)
\def\xref{\futurelet\next\xrefswitch} % branch on the next character
\def\xrefswitch{\begingroup\ifx\next|\aftergroup\vxref
  \else\ifx\next\<\aftergroup\anglexref
    \else\aftergroup\normalxref \fi\fi \endgroup}
\def\vxref|{\catcode`\\=\active \futurelet\next\vxrefswitch}
\def\vxrefswitch#1|{\catcode`\\=0
  \ifx\next\empty\def\xreftype{2}%
    \def\next{{\tt\bslash\text}}% code 2, |\arg|
  \else\def\xreftype{1}\def\next{{\tt\text}}\fi % code 1, |arg|
  \edef\text{#1}\makexref}
{\catcode`\|=0 \catcode`\\=\active |gdef\{}}
\def\anglexref\<#1>{\def\xreftype{3}\def\text{#1}%
  \def\next{\<\text>}\makexref} % code 3, \<arg>
\def\normalxref#1{\def\xreftype{0}\def\text{#1}\let\next=\text\makexref}
```

Указатель не составляется, когда переключатель `proofmode` установлен на истину, поскольку материал для указателя собирается только во время пробных прогонов, а не в том триумфальном случае, когда книга окончательно печатается.

```
\newif\ifproofmode
\proofmodetrue % this should be false when making camera-ready copy
\def\makexref{\ifproofmode\insert\margin{\hbox{\marginstyle\text}}%
   \xdef\writeit{\write\inx{\text\space!\xreftype\space
     \noexpand\number\pageno.}}\writeit
   \else\ifhmode\kern0pt \fi\fi
  \ifsilent\ignorespaces\else\next\fi}
```

(`\insert` подавляет перенос, когда проверяются корректуры, поэтому испускается `\kern0pt`, чтобы обеспечить согласующееся поведение в противном случае.)

Материал, который накапливается в файле `index.tex`, дает хорошее первое приближение указателя, но не содержит достаточно информации для того, чтобы выполнить всю работу. Предмет обсуждения часто встречается на нескольких страницах, а в файле обычно указывается только первая из них. Автор предпочел не генерировать указатель автоматически. Ему нравится перечитать свою книгу, проверяя перекрестные ссылки, тем самым имея удобный случай еще

раз все обдумать и выловить, пока еще не поздно, разнообразные ошибки. В результате книга задерживается, но указатель становится лучше. Поэтому он создал в `manmac`'е такую схему составления указателя, что она обеспечивает только ключи для составления реального указателя. С другой стороны, можно было бы расширить указанные выше макрокоманды и достигнуть исчерпывающей системы, которая генерирует отличный указатель без всякого последующего вмешательства; см., например, "An indexing facility for TEX" , Terry Winograd и Bill Paxton, в *TUGboat* **1** (1980), A1–A12.

Макрокоманды формата `manmac` представлены теперь полностью. Мы завершим это приложение еще одним примером их использования. Глава 27 упоминает, что желательно создавать длинную книгу по маленьким частям, используя файл "гранки". Автор применил эту стратегию для *The TEXbook*, вводя каждую главу в маленький файл `galley.tex`, который выглядит так:

```
\input manmac
\tenpoint
\pageno=800
\def\rhead{Experimental Pages for The \TeX book}
\def\chapno{ X}
{\catcode`\%=12 \immediate\write\ans{% Answers for galley proofs:}}
```

⋮

⟨новый проверяемый текст, обычно, целая глава⟩

⋮

```
% that blank line will stop an unfinished \answer
\immediate\closeout\ans
\vfill\eject
\ninepoint \input answers % typeset the new answers, if any
\bye
```

Гораздо легче использовать макрокоманды,
чем определять их.
...
Использование библиотеки макрокоманд
довольно точно копирует использование
библиотеки подпрограмм
в языках программирования.
Образуются те же уровни
специализации, от подпрограмм
общего пользования до специфических
подпрограмм внутри отдельной программы,
и имеется такая же потребность
в программисте с особым умением
создавать подпрограммы.

It is much easier to use macros
than to define them.
...
The use of macro libraries, in fact,
mirrors almost exactly the use
of subroutine libraries
for programming languages.
There are the same levels
of specialization, from publicly
shared subroutines to special
subroutines within a single program,
and there is the same need
for a programmer with particular
skills to define the subroutines.

— PETER BROWN, *Macro Processors* (1974)

Эпиграф является одним из самых
приятных научных обычаев.
Работа Дональда Кнута по фундаментальным
алгоритмам осталась бы столь же важной, если
бы она и не начиналась цитатой из Бетти Крокера,
но не доставила бы столько удовольствия.
Одним из развлечений с эпиграфами является
использование источника для неожиданных целей.

The epigraph is among the most
delightful of scholarly habits.
Donald Knuth's work on fundamental
algorithms would be just as important
if he hadn't begun with a quotation
from Betty Crocker, but not so enjoyable.
Part of the fun of an epigraph is turning
a source to an unexpected use.

— MARY-CLAIRE VAN LEUNEN, *A Handbook for Scholars* (1978)

F

Таблицы шрифтов

Это приложение делает обзор основных характеристик Computer Modern шрифтов. TEX может набирать документы любыми шрифтами с любым оформлением символов. Здесь описаны шрифты и макеты, которые соответствуют формату начального TEX'а, то есть макрокомандам из приложения B. (Полную информацию о семействе Computer Modern шрифтов, включающую программы METAFONT'а, которые рисуют символы, можно найти в книге автора *Computer Modern Typefaces*.)

В начале приложения показано содержание шрифтов, а в конце — вид символов в математических формулах. (Соглашения, которые применяются в нематематическом тексте, см. в приложении B.)

В каждом из Computer Modern шрифтов содержится 128 символов, хотя TEX может иметь до 256 символов на шрифт. Макеты текстовых шрифтов показаны в таблице, которая иллюстрирует шрифт `cmr10` (Computer Modern Roman 10 пунктов). Так, например, если вы запрашиваете `\char'35` в шрифте `cmr10`, то получаете символ Æ. Эти текстовые шрифты включают лигатуры и акценты, описанные в главе 9. Каждый видимый символ ASCII располагается в своей позиции ASCII. Некоторые символы ASCII (а именно, `" < > \ _ { | }`) не включены, поскольку они не встречаются в обычных шрифтах. Если вы по ошибке введете `''`, то получите ", а `<` вне математической моды дает ¡! Кстати, десять цифр имеют ширину `0.5em`.

Рисунок 1. Макет текстового шрифта `cmr10` (`\rm`, `\textfont0`).

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*	
'00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	"0x
'01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	
'02x	ı	ȷ	`	´	ˇ	˘	¯	°	"1x
'03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
'04x	‘	!	”	#	$	%	&	’	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	¡	=	¿	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	“	]	^	˙	
'14x	‘	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	–	—	˝	~	¨	
	"8	"9	"A	"B	"C	"D	"E	"F	

Начальный TEX использует шестнадцать основных шрифтов:

`cmr10`	(Computer Modern Roman в 10 пунктов)	текст.
`cmr7`	(Computer Modern Roman в 7 пунктов)	
`cmr5`	(Computer Modern Roman в 5 пунктов)	
`cmbx10`	(Computer Modern Bold Extended в 10 пунктов)	
`cmbx7`	(Computer Modern Bold Extended в 7 пунктов)	
`cmbx5`	(Computer Modern Bold Extended в 5 пунктов)	
`cmsl10`	(Computer Modern Slanted Roman в 10 пунктов)	
`cmti10`	(Computer Modern Text Italic в 10 пунктов)	
`cmtt10`	(Computer Modern Typewriter type в 10 пунктов)	спец.
`cmmi10`	(Computer Modern Math Italic в 10 пунктов)	
`cmmi7`	(Computer Modern Math Italic в 7 пунктов)	
`cmmi5`	(Computer Modern Math Italic в 5 пунктов)	
`cmsy10`	(Computer Modern Math Symbols в 10 пунктов)	
`cmsy7`	(Computer Modern Math Symbols в 7 пунктов)	
`cmsy5`	(Computer Modern Math Symbols в 5 пунктов)	
`cmex10`	(Computer Modern Math Extension в 10 пунктов)	

У первых восьми из них одинаковый макет, но для `cmr5` не нужны лигатуры, а у многих символов `cmti10` другая форма. Например, амперсанд превращается в “E.T.”, а доллар в фунт стерлингов:

Рисунок 2. Макет текстового шрифта `cmti10` (`\it`).

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
´01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	
´02x	ı	ȷ	`	´	ˇ	˘	¯	°	″1x
´03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	‘	!	”	#	£	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	¡	=	¿	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	“	]	^	˙	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	–	—	″	˜	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

Шрифт пишущей машинки `cmtt10` похож на обычный шрифт, но включает все видимые символы ASCII в их правильном положении. Он также имеет вертикальные стрелки ↑ и ↓ и знак одиночной кавычки '. По сравнению с текстовыми макетами в этом шрифте изменены четырнадцать из 128 позиций, а именно, коды *´013–´017*, *´040*, *´042*, *´074*, *´076*, *´134*, *´137* и *´173–´175*. Лигатуры отсутствуют, кроме испанских ¡ и ¿. (Символы испанских лигатур появляются в других позициях, но для пользователя здесь нет отличия, поскольку каждый шрифт сообщает TEX'у расположение своих лигатур.) Польская ł, ṫочечный акцент и длинный венгерский ű́мляут исчезли, уступив место новым символам. Позиции *´052* и *´055* также слегка отличаются от нормальных соглашений. Звездочка поднята не так высоко, как обычно, а знак тире совпадает с минусом.

Каждый символ в `cmtt10` имеет одинаковую ширину `0.5em`; пробелы между словами также одинаковые и не могут ни растягиваться, ни сжиматься. При наборе шрифтом пишущей машинки TEX в конце предложения ставит по два пробела. (Соглашения по пробелам можно изменить, давая значения `\spaceskip` и `\xspaceskip`, или можно дать новые значения параметрам `\fontdimen`, которые были кратко описаны.)

Рисунок 3. Макет шрифта пишущей машинки `cmtt10` (`\tt`).

	´0	*´1*	*´2*	*´3*	*´4*	*´5*	*´6*	*´7*	
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
´01x	Φ	Ψ	Ω	↑	↓	'	¡	¿	
´02x	ı	ȷ	`	´	˘	¯	˚	¸	″1x
´03x	˛	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	␣	!	"	#	$	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	<	=	>	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	\	]	^	_	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	{	|	}	~	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

С первого взгляда видно, что математический курсив `cmmi10` сильно отличается от текстового курсива. Он содержит малые и большие греческие буквы. Это греческие буквы математики, а не текстового шрифта для набора классической греческой литературы. А если рассмотреть не греческие буквы, то видно, что пропорции букв и пробелы сильно изменены по сравнению с `cmti10`, чтобы они лучше выглядели в математической моде.

В позициях *′050–′077* и *′133–′137* появляются специальные ненаклонные символы, включающие цифры "старого стиля":и `$\mit1984$`, и `$\oldstyle1984$` дают 1984. Некоторые символы предназначены, чтобы комбинироваться с другими: например, *′054* — это начало символа '↪'. (См. определение `\hookrightarrow` в приложении B.) Эта часть шрифта не заслуживает имени математического курсива, это просто место для символов, которые не поместились где-нибудь еще. (Автор не хотел оставлять свободные места, поскольку это соблазняло бы людей заполнять их несовместимым способом.)

Начальный TEX берет запятые, точки и наклонные черты из `cmmi10` в математической моде, так что в формулах делается подходящее кернирование, без чего в них недоставало бы пробелов. Для правильного положения акцентов в этом шрифте надо установить `\skewchar` в *′177*.

Рисунок 4. Макет математического курсива `cmmi10` (`\mit`, `\textfont1`).

	′0	*′1*	*′2*	*′3*	*′4*	*′5*	*′6*	*′7*	
′00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	"0x
′01x	Φ	Ψ	Ω	α	β	γ	δ	ϵ	
′02x	ζ	η	θ	ι	κ	λ	μ	ν	"1x
′03x	ξ	π	ρ	σ	τ	υ	ϕ	χ	
′04x	ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ	"2x
′05x	$\leftharpoonup$	$\leftharpoondown$	$\rightharpoonup$	$\rightharpoondown$	‘	’	$\triangleright$	$\triangleleft$	
′06x	0	1	2	3	4	5	6	7	"3x
′07x	8	9	.	,	<	/	>	$\star$	
′10x	∂	A	B	C	D	E	F	G	"4x
′11x	H	I	J	K	L	M	N	O	
′12x	P	Q	R	S	T	U	V	W	"5x
′13x	X	Y	Z	$\flat$	$\natural$	$\sharp$	$\smile$	$\frown$	
′14x	ℓ	a	b	c	d	e	f	g	"6x
′15x	h	i	j	k	l	m	n	o	
′16x	p	q	r	s	t	u	v	w	"7x
′17x	x	y	z	$\imath$	$\jmath$	$\wp$	$\vec{}$	$\frown$	
	"8	"9	"A	"B	"C	"D	"E	"F	

Когда TEX набирает математику, он предполагает, что семейство 0 содержит нормальные романские шрифты, а семейство 1, 2 и 3 содержит математические курсивные, математические символьные и математические расширенные шрифты. Специальным символам этих шрифтов обычно даны символические имена инструкцией `\mathchardef`, которая присваивает символу шестнадцатиричный код. Этот код состоит из четырех цифр, где первая указывает, какой вид символа подразумевается, вторая задает семейство, а две другие дают позицию. Например,

```
\mathchardef\ll="321C
```

говорит, что `\ll` — это символ `"1C` в математическом символьном шрифте (семейство 2), и что это "отношение" (класс 3). Полный список символических имен начального TEX'а приведен в этом приложении позднее.

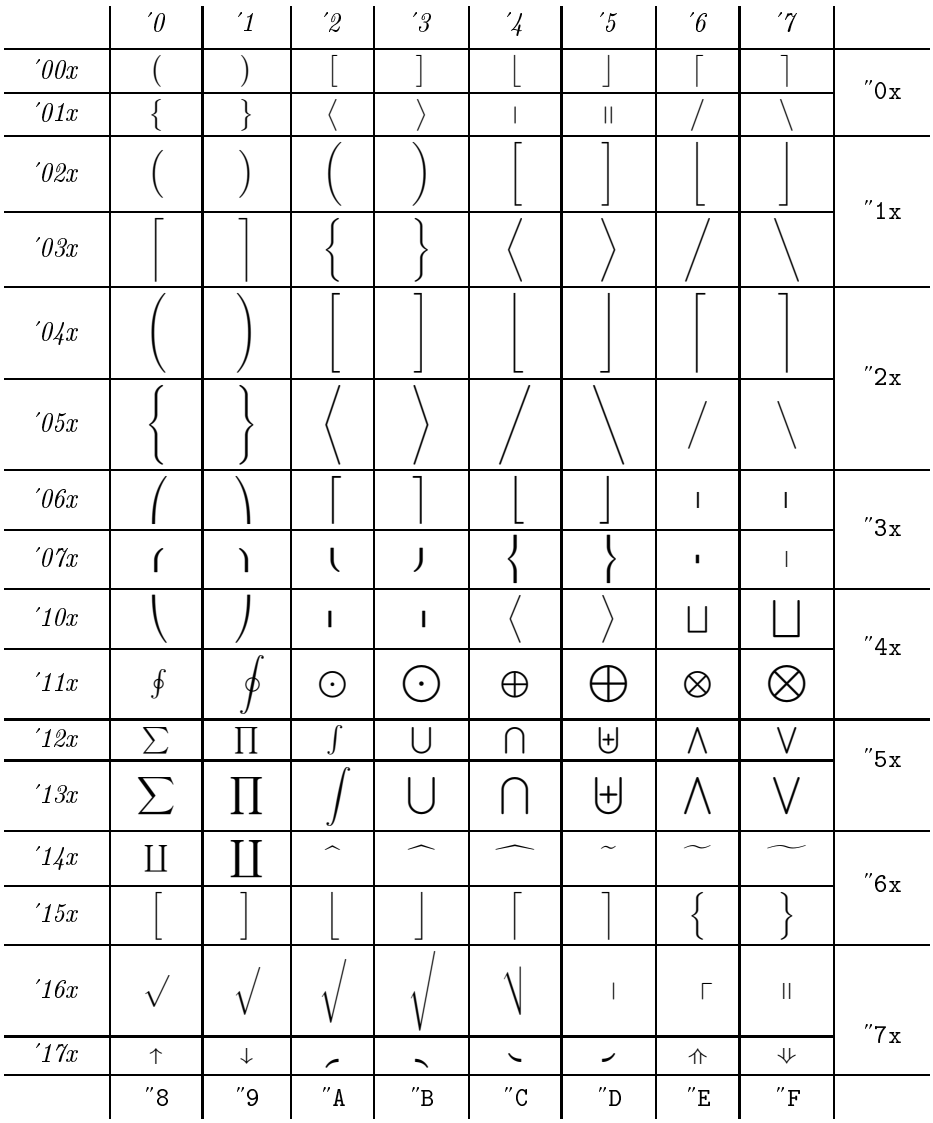
Шрифт `cmsy10` — это математический символьный шрифт начального TEX'а. Он содержит 128 символов, как показано ниже. Его `\skewchar` должно быть установлено в *'060*, так что математические акценты будут правильно расположены над каллиграфическими заглавными буквами.

Рисунок 5. Макет шрифта математических символов `cmsy10` (`\cal`, `\textfont2`).

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*	
'00x	$-$	$\cdot$	$\times$	$*$	$\div$	$\diamond$	$\pm$	$\mp$	`"0x`
'01x	$\oplus$	$\ominus$	$\otimes$	$\oslash$	$\odot$	$\bigcirc$	$\circ$	$\bullet$	
'02x	$\asymp$	$\equiv$	$\subseteq$	$\supseteq$	$\leq$	$\geq$	$\preceq$	$\succeq$	`"1x`
'03x	$\sim$	$\approx$	$\subset$	$\supset$	$\ll$	$\gg$	$\prec$	$\succ$	
'04x	$\leftarrow$	$\rightarrow$	$\uparrow$	$\downarrow$	$\leftrightarrow$	$\nearrow$	$\searrow$	$\simeq$	`"2x`
'05x	$\Leftarrow$	$\Rightarrow$	$\Uparrow$	$\Downarrow$	$\Leftrightarrow$	$\nwarrow$	$\swarrow$	$\propto$	
'06x	$\prime$	∞	$\in$	$\ni$	$\triangle$	$\triangledown$	$/$	$\flat$	`"3x`
'07x	$\forall$	$\exists$	$\neg$	$\emptyset$	$\Re$	$\Im$	$\top$	$\bot$	
'10x	$\aleph$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$	`"4x`
'11x	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$	
'12x	$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	`"5x`
'13x	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	$\cup$	$\cap$	$\uplus$	$\wedge$	$\vee$	
'14x	$\vdash$	$\dashv$	$\lfloor$	$\rfloor$	$\lceil$	$\rceil$	$\{$	$\}$	`"6x`
'15x	$\langle$	$\rangle$	$\mid$	$\parallel$	$\updownarrow$	$\Updownarrow$	$\backslash$	$\wr$	
'16x	$\surd$	$\amalg$	∇	$\int$	$\sqcup$	$\sqcap$	$\sqsubseteq$	$\sqsupseteq$	`"7x`
'17x	$\S$	$\dagger$	$\ddagger$	$\P$	$\clubsuit$	$\diamondsuit$	$\heartsuit$	$\spadesuit$	
	`"8`	`"9`	`"A`	`"B`	`"C`	`"D`	`"E`	`"F`	

Последний шрифт начального TEX' — это `cmex10`, который содержит большие символы и части, которые можно использовать в еще больших символах. Например, можно сделать произвольно большую левую круглую скобку, поместив сверху *´060*, снизу *´100*, а в середине — сколько угодно копий *´102*. Большие квадратные корни делаются из *´164*, *´165* и *´166*, а большие левые фигурные скобки из четырех частей: *´070*, *´072*, *´074*, *´076*.

Рисунок 6. Математический расширенный шрифт `cmex10` (`\textfont3`).



Когда TEX "загружает" шрифт в память, он не рассматривает реальные формы символов, а загружает только *метрическую информацию шрифта* (например, `cmr10.tfm`), которая содержит информацию о высоте, ширине, глубине и курсивной поправке символов вместе с информацией о лигатурах и кернах. Более того, метрическая информация, которая поступает с шрифтом типа `cmex10`, говорит TEX'у, что некоторые символы образуют серии. Например, все левые круглые скобки связаны вместе в порядке увеличения размера: *'000*, *'020*, *'022* , *'040*, за которыми следуют расширяемые левые круглые скобки, представляющие собой $'060 + ['102]^n + '100$. Аналогично, два знака суммы (*'120*, *'130*) и три акцента `\widehat` (*'142*, *'143*, *'144*) связаны вместе. Приложение G объясняет, как TEX выбирает конкретные размеры для математических операторов и математических акцентов.

У каждого шрифта есть не меньше семи параметров `\fontdimen`, которые имеют следующие значения и типичные величины (округленные до двух десятичных знаков):

#	Смысл	Значение в	`cmr10`	`cmbx10`	`cmsl10`	`cmti10`	`cmtt10`	`cmmi10`
1	наклон на pt		0.00 pt	0.00 pt	0.17 pt	0.25 pt	0.00 pt	0.25 pt
2	пробел между словами		3.33 pt	3.83 pt	3.33 pt	3.58 pt	5.25 pt	0.00 pt
3	его растяжимость		1.67 pt	1.92 pt	1.67 pt	1.53 pt	0.00 pt	0.00 pt
4	его сжимаемость		1.11 pt	1.28 pt	1.11 pt	1.02 pt	0.00 pt	0.00 pt
5	x-высота		4.31 pt	4.44 pt	4.31 pt	4.31 pt	4.31 pt	4.31 pt
6	ширина квадрата		10.00 pt	11.50 pt	10.00 pt	10.22 pt	10.50 pt	10.00 pt
7	добавочный пробел		1.11 pt	1.28 pt	1.11 pt	1.02 pt	5.25 pt	0.00 pt

Наклон используется для размещения акцентов. Следующие три параметра определяют промежутки между словами при наборе текста. Следующие два определяют связанные со шрифтом размеры `1ex` и `1em`, а последний — это дополнительная величина, которая добавляется к пробелам между словами в конце предложений (т.е., когда `\spacefactor` равен 2000 или более, а `\xspaceskip` равен нулю). Когда шрифт увеличивается (с помощью `at` или `scaled`), все параметры, кроме наклона, увеличиваются одновременно с шрифтом, загруженным в память TEX'а.

Заметим, что в `cmmi10` пробелы равны нулю. Это признак шрифта, который используется только для набора математики. Правила в приложении G утверждают, что между соседними символами в таких шрифтах добавляется курсивная поправка.

Математическим символьным шрифтам (то есть шрифтам семейства 2) требуется не меньше 22 параметров `\fontdimen` вместо обычных семи. Аналогично, в математическом расширенном шрифте этих параметров должно быть как минимум 13. Смысл этих дополнительных параметров объясняется в приложении G. Если вы хотите увеличить число параметров сверх числа, которое находится в файле метрической информации шрифта, то можете присвоить новые значения сразу после того, как этот шрифт

загружен. Например, если некоторый шрифт `\ff` с семью параметрами только что введен в память TEX'а, команда `\fontdimen13\ff=5pt` установит параметр номер 13 в 5 pt. Промежуточные параметры с номерами 8–12 будут установлены в ноль. Можно даже задать более семи параметров для `\nullfont`, имея в виду, что значения присваиваются до того, как какой-нибудь шрифт будет реально загружен.

Теперь, когда показаны макеты всех шрифтов, настало время рассмотреть имена разных математических символов. Начальный TEX определяет более 200 команд, которыми можно ссылаться на математические символы без поиска их в макетах. Обычно удобно вызывать символы по их именам. В этом случае можно легко адаптировать свою рукопись к другим шрифтам, к тому же она будет намного читабельнее.

Символы естественным образом подразделяются на группы по их математическому классу (Ord, Op, Bin, Rel, Open, Close или Punct), поэтому будем следовать порядку, в котором мы их обсуждали. N.B.: если не утверждается обратное, математические символы доступны только в математической моде. Например, если вы скажете `\alpha` в горизонтальной моде, TEX сообщит об ошибке и попытается вставить знак `$`.

1. Строчные греческие буквы.

α `\alpha`	ι `\iota`	ϱ `\varrho`
β `\beta`	κ `\kappa`	σ `\sigma`
γ `\gamma`	λ `\lambda`	ς `\varsigma`
δ `\delta`	μ `\mu`	τ `\tau`
ϵ `\epsilon`	ν `\nu`	υ `\upsilon`
ε `\varepsilon`	ξ `\xi`	ϕ `\phi`
ζ `\zeta`	o `o`	φ `\varphi`
η `\eta`	π `\pi`	χ `\chi`
θ `\theta`	ϖ `\varpi`	ψ `\psi`
ϑ `\vartheta`	ρ `\rho`	ω `\omega`

Здесь нет `\omicron`, поскольку она выглядит так же, как `o`. Заметим, что буква `\upsilon` (υ) здесь чуть шире, чем `v` (v); и ту, и другую следует отличать от `\nu` (ν). Аналогично, `\varsigma` (ς) не следует путать с `\zeta` (ζ). Оказывается, что `\varsigma` и `\upsilon` никогда не используются в математических формулах. Они включены в начальный TEX только потому, что оказались нужны в коротких греческих цитатах (ср. приложение J).

2. Заглавные греческие буквы.

Γ `\Gamma`	Ξ `\Xi`	Φ `\Phi`
Δ `\Delta`	Π `\Pi`	Ψ `\Psi`
Θ `\Theta`	Σ `\Sigma`	Ω `\Omega`
Λ `\Lambda`	Υ `\Upsilon`	

Остальные греческие заглавные буквы печатаются из романского алфавита

(`\Alpha` ≡ `{\rm A}`, `\Beta` ≡ `{\rm B}`, и т.д). Принято использовать ненаклонные буквы для заглавных греческих букв, а наклонные — для строчных, но можно получать ($\Gamma, \Delta, \ldots, \Omega$) и вводя `$({\mit\Gamma}, {\mit\Delta}, \ldots, {\mit\Omega})$`.

3. Каллиграфические заглавные буквы. Чтобы получить буквы $\mathcal{A} \ldots \mathcal{Z}$, которые приведены на рисунке 5, вводите `${\cal A}\ldots{\cal Z}$`. Некоторые другие алфавиты также используются с математикой (особенно ломанный, рукописный и "blackboard bold"). Они не входят в начальный TEX, но более изысканные форматы типа $\mathcal{AMS}$-TEX включают их.

4. Смешанные символы типа Ord.

$\aleph$ `\aleph`	$\prime$ `\prime`	$\forall$ `\forall`
$\hbar$ `\hbar`	$\emptyset$ `\emptyset`	$\exists$ `\exists`
$\imath$ `\imath`	∇ `\nabla`	$\neg$ `\neg`
$\jmath$ `\jmath`	$\surd$ `\surd`	$\flat$ `\flat`
ℓ `\ell`	$\top$ `\top`	$\natural$ `\natural`
$\wp$ `\wp`	$\bot$ `\bot`	$\sharp$ `\sharp`
$\Re$ `\Re`	$\|$ `\\|`	$\clubsuit$ `\clubsuit`
$\Im$ `\Im`	$\angle$ `\angle`	$\diamondsuit$ `\diamondsuit`
∂ `\partial`	$\triangle$ `\triangle`	$\heartsuit$ `\heartsuit`
∞ `\infty`	$\backslash$ `\backslash`	$\spadesuit$ `\spadesuit`

Бесточечные буквы `\imath` и `\jmath` используются, когда к i и j добавляется акцент. Например, `$\hat\imath$` дает $\hat\imath$. Символ `\prime` предназначен для использования в верхних и нижних индексах, как это объяснялось в главе 16, так чтобы обычно он встречается в уменьшенном размере. Символ же `\angle` составлен из кусочков и, когда появляется в верхнем или нижнем индексе, не уменьшается.

5. Цифры. Для цифр курсива *0123456789* надо сказать `{\it0123456789}`, для жирных цифр **0123456789** — `{\bf0123456789}`, а для цифр старого стиля 0123456789 — `{\oldstyle0123456789}`. Эти соглашения действуют также и вне математической моды.

6. "Большие" операторы. Следующие символы могут быть в двух размерах, для текстового и выделенного стилей:

$\sum \sum$ `\sum`	$\cap \bigcap$ `\bigcap`	$\odot \bigodot$ `\bigodot`
$\prod \prod$ `\prod`	$\cup \bigcup$ `\bigcup`	$\otimes \bigotimes$ `\bigotimes`
$\coprod \coprod$ `\coprod`	$\sqcup \bigsqcup$ `\bigsqcup`	$\oplus \bigoplus$ `\bigoplus`
$\int \int$ `\int`	$\vee \bigvee$ `\bigvee`	$\uplus \biguplus$ `\biguplus`
$\oint \oint$ `\oint`	$\wedge \bigwedge$ `\bigwedge`	

Важно отметить отличие этих больших символов Op от похожих, но меньших символов Bin, у которых такие же имена, за исключением приставки `big`. Большие операторы обычно встречаются в начале формулы или подформулы и обычно имеют индексы, а бинарные операции обычно встречаются между двумя символами или подформулами и редко имеют индексы. Например,

`$\bigcup_{n=1}^m(x_n\cup y_n)$` дает $\bigcup_{n=1}^m(x_n\cup y_n)$

Большие операторы `\sum`, `\prod`, `\coprod` и `\int` также надо отличать от символов `\Sigma` (Σ), `\Pi` (Π), `\amalg` ($\amalg$) и `\smallint` ($\smallint$), соответственно. Оператор `\smallint` используется редко.

7. Бинарные операции. Кроме + и −, можно вводить

$\pm$ `\pm`	$\cap$ `\cap`	$\vee$ `\vee`
$\mp$ `\mp`	$\cup$ `\cup`	$\wedge$ `\wedge`
$\setminus$ `\setminus`	$\uplus$ `\uplus`	$\oplus$ `\oplus`
$\cdot$ `\cdot`	$\sqcap$ `\sqcap`	$\ominus$ `\ominus`
$\times$ `\times`	$\sqcup$ `\sqcup`	$\otimes$ `\otimes`
$*$ `\ast`	$\triangleleft$ `\triangleleft`	$\oslash$ `\oslash`
$\star$ `\star`	$\triangleright$ `\triangleright`	$\odot$ `\odot`
$\diamond$ `\diamond`	$\wr$ `\wr`	$\dagger$ `\dagger`
$\circ$ `\circ`	$\bigcirc$ `\bigcirc`	$\ddagger$ `\ddagger`
$\bullet$ `\bullet`	$\bigtriangleup$ `\bigtriangleup`	$\amalg$ `\amalg`
$\div$ `\div`	$\bigtriangledown$ `\bigtriangledown`	

Принято говорить `$G\backslash H$` для двойного косета G по H ($G\backslash H$) и `$p\backslash n$` для обозначения, что p делится на n ($p\backslash n$). Но `$X\setminus Y$` обозначает элементы множества X минус элементы множества Y ($X\setminus Y$). Обе операции используют один и тот же символ, но `\backslash` имеет тип Ord, а `\setminus` — тип Bin (поэтому TEX помещает вокруг него большие пробелы).

8. Отношения. Кроме <, > и =, можно вводить

$\leq$ `\leq`	$\geq$ `\geq`	$\equiv$ `\equiv`
$\prec$ `\prec`	$\succ$ `\succ`	$\sim$ `\sim`
$\preceq$ `\preceq`	$\succeq$ `\succeq`	$\simeq$ `\simeq`
$\ll$ `\ll`	$\gg$ `\gg`	$\asymp$ `\asymp`
$\subset$ `\subset`	$\supset$ `\supset`	$\approx$ `\approx`
$\subseteq$ `\subseteq`	$\supseteq$ `\supseteq`	$\cong$ `\cong`
$\sqsubseteq$ `\sqsubseteq`	$\sqsupseteq$ `\sqsupseteq`	$\bowtie$ `\bowtie`
$\in$ `\in`	$\ni$ `\ni`	$\propto$ `\propto`
$\vdash$ `\vdash`	$\dashv$ `\dashv`	$\models$ `\models`
$\smile$ `\smile`	$\mid$ `\mid`	$\doteq$ `\doteq`
$\frown$ `\frown`	$\parallel$ `\parallel`	$\perp$ `\perp`

Символы `\mid` и `\parallel` определяют отношения, которые используют те же символы, которые вы получаете из `|` и `\|`. Когда эти символы являются отношениями, TEX помещает вокруг них пробелы.

9. Отношения с отрицанием. Многие из только что перечисленных отношений могут быть отрицательными или "перечеркнутыми", если перед ними поставить приставку `\not` следующим образом:

$\not<$ `\not<`	$\not>$ `\not>`	$\not=$ `\not=`
$\not\leq$ `\not\leq`	$\not\geq$ `\not\geq`	$\not\equiv$ `\not\equiv`
$\not\prec$ `\not\prec`	$\not\succ$ `\not\succ`	$\not\sim$ `\not\sim`
$\not\preceq$ `\not\preceq`	$\not\succeq$ `\not\succeq`	$\not\simeq$ `\not\simeq`
$\not\subset$ `\not\subset`	$\not\supset$ `\not\supset`	$\not\approx$ `\not\approx`
$\not\subseteq$ `\not\subseteq`	$\not\supseteq$ `\not\supseteq`	$\not\cong$ `\not\cong`
$\not\sqsubseteq$ `\not\sqsubseteq`	$\not\sqsupseteq$ `\not\sqsupseteq`	$\not\asymp$ `\not\asymp`

Символ `\not` является символом отношения нулевой ширины, так что он будет перекрывать отношение, которое следует за ним. Его положение не всегда идеально, поскольку некоторые символы отношения шире других. Например, `\not\in` дает $\not\in$, но лучше иметь более крутое зачеркивание $\notin$. Последний символ получается специальной командой `\notin`. Определение `\notin` в приложении B показывает, как можно создавать похожие символы.

10. Стрелки. Существует большой класс указывающих отношений:

$\leftarrow$ `\leftarrow`	$\longleftarrow$ `\longleftarrow`	$\uparrow$ `\uparrow`
$\Leftarrow$ `\Leftarrow`	$\Longleftarrow$ `\Longleftarrow`	$\Uparrow$ `\Uparrow`
$\rightarrow$ `\rightarrow`	$\longrightarrow$ `\longrightarrow`	$\downarrow$ `\downarrow`
$\Rightarrow$ `\Rightarrow`	$\Longrightarrow$ `\Longrightarrow`	$\Downarrow$ `\Downarrow`
$\leftrightarrow$ `\leftrightarrow`	$\longleftrightarrow$ `\longleftrightarrow`	$\updownarrow$ `\updownarrow`
$\Leftrightarrow$ `\Leftrightarrow`	$\Longleftrightarrow$ `\Longleftrightarrow`	$\Updownarrow$ `\Updownarrow`
$\mapsto$ `\mapsto`	$\longmapsto$ `\longmapsto`	$\nearrow$ `\nearrow`
$\hookleftarrow$ `\hookleftarrow`	$\hookrightarrow$ `\hookrightarrow`	$\searrow$ `\searrow`
$\leftharpoonup$ `\leftharpoonup`	$\rightharpoonup$ `\rightharpoonup`	$\swarrow$ `\swarrow`
$\leftharpoondown$ `\leftharpoondown`	$\rightharpoondown$ `\rightharpoondown`	$\nwarrow$ `\nwarrow`
$\rightleftharpoons$ `\rightleftharpoons`		

Стрелки будут увеличиваться, как и ограничители (см. главу 17). Чтобы поместить символы над стрелкой, начальный TEX имеет макрокоманду `\buildrel`: вы вводите `\buildrel`⟨верхний индекс⟩`\over`⟨отношение⟩, и верхний индекс помещается сверху отношения, так же как пределы помещаются над большими операторами. Например,

$\xrightarrow{\alpha\beta}$ `\buildrel \alpha\beta \over \longrightarrow`

$\overset{\text{def}}{=}$ `\buildrel \rm def \over =`

(В этом контексте `\over` не определяет дробь.)

11. Открывающие. Кроме (, доступны следующие левые ограничители

$[$	`\lbrack`	$\lfloor$	`\lfloor`	$\lceil$	`\lceil`
$\{$	`\lbrace`	$\langle$	`\langle`		

Можно вводить просто `[`, чтобы получить `\lbrack`. Все эти ограничители будут увеличены, если поставить перед ними `\bigl`, `\Bigl`, `\biggl`, `\Biggl` или `\left`. Глава 17 также упоминает `\lgroup` и `\lmoustache`, которые доступны в размерах, больших `\big`. Если вам нужно больше ограничителей, в нормальном текстовом размере достаточно хорошо работают следующие комбинации:

$[\![$	`\lbrack\!\lbrack`	$\langle\!\langle$	`\langle\!\langle`	$(($	`(\!(`

12. Закрывающие. Также имеются соответствующие правые ограничители:

$]$	`\rbrack`	$\rfloor$	`\rfloor`	$\rceil$	`\rceil`
$\}$	`\rbrace`	$\rangle$	`\rangle`		

Все, что работает для открывающих символов, также работает для закрывающих, но перевернуто.

13. Пунктуация. После запятых и точек с запятой, которые встречаются в математических формулах, TEX помещает тонкий пробел. То же самое он делает и для двоеточия, которое вызывается командой `\colon`. (В других случаях двоеточие считается отношением, как в $x := y$ и в $a : b :: c : d$, которые вводятся как `$x:=y$` и `$a:b::c:d$`.) Приведем примеры `\colon`

$f\colon A \to B$	`$f\colon A\rightarrow B$`
$L(a, b; c\colon x, y; z)$	`$L(a,b;c\colon x,y;z)$`

Начальный TEX также определяет, чтобы `\ldotp` и `\cdotp` были “.” и “·” с теми же пробелами, как при запятых и при точках с запятыми. Эти символы не встречаются в формулах, но они полезны в определениях `\ldots` и `\cdots`.

14. Альтернативные имена. Если вам не нравятся имена начального TEX'а для некоторых математических символов — например, если вам легче запомнить другое имя — есть простое средство. Вы просто говорите, например, `\let\cupcap=\asymp`, и затем можете вводить `f(n)\cupcap n` вместо `f(n)\asymp n`.

Некоторые символы имеют альтернативные имена, которые так широко используются, что начальный TEX имеет для них по две и более команды:

$\neq$	`\ne` или `\neq`	(то же, что `\not=`)
$\leq$	`\le`	(то же, что `\leq`)
$\geq$	`\ge`	(то же, что `\geq`)

{	`\{`	(то же, что `\lbrace`)
}	`\}`	(то же, что `\rbrace`)
→	`\to`	(то же, что `\rightarrow`)
←	`\gets`	(то же, что `\leftarrow`)
∋	`\owns`	(то же, что `\ni`)
∧	`\land`	(то же, что `\wedge`)
∨	`\lor`	(то же, что `\vee`)
¬	`\lnot`	(то же, что `\neg`)
|	`\vert`	(то же, что `|`)
‖	`\Vert`	(то же, что `\\|`)

Также имеется `\iff` (⟺), почти такой же как `\Longleftrightarrow`, но помещающий по краям по одному пробелу.

15. Нематематические символы. Начальный TEX делает четыре специальные символа доступными вне математической моды, хотя они и берутся из математического шрифта:

§ `\S`
¶ `\P`
† `\dag`
‡ `\ddag`

Эти команды ведут себя не как обычные математические символы: они не изменяют свой размер, когда появляются в индексах, а когда вы используете их в формулах, то должны, например, сказать, `$x^{\P}$` вместо `$x^\P$`. Однако, символы `\dag` и `\ddag` доступны в математической моде под именами `\dagger` и `\ddagger`. Легко определить и математические эквиваленты `\S` и `\P`, если эти символы вдруг возникнут в фантазии математика.

Не ищи к колодцам дальний путь,
а попробуй рядом зачерпнуть.

Seek not for fresher founts afar,
Just drop your bucket where you are.

— SAM WALTER FOSS, *Back Country Poems* (1892)

Никакой наборщик не будет иметь
всех возможных знаков и символов.
Количество специальных знаков
и символов почти беспредельно,
и все время вводятся все
новые и новые

No one compositor will have all
the signs and symbols available.
The number of special signs
and symbols is almost limitless,
with new ones being introduced
all the time.

— UNIVERSITY OF CHICAGO PRESS, *Manual of Style* (1969)

G

Генерация боксов из формул

Тем, кто определяет новые математические шрифты и/или макрокоманды, иногда полезно знать, как TEX обращается с составными частями формул. Цель этого приложения — определить точные позиционные правила, по которым TEX преобразует математический список в горизонтальный. (Есть хорошая идея — перед тем, как читать дальше, просмотрим введение в математические списки в главе 17. Во всем этом приложении подразумевается "двойной знак опасного поворота")

TEX, когда печатает формулы, полагается на множество параметров, и можно изменить любой из них, или даже все. Но, конечно же, прежде чем менять что-то, надо узнать, что означает каждый параметр. Поэтому у каждого из приведенного ниже правила есть номер, а таблица в конце приложения показывает, какой параметр от какого правила зависит.

Наиболее важные параметры относятся к символьным (семейство 2) и к расширенным (семейство 3) шрифтам. TEX не будет печатать формулу, если только каждый из `\textfont2`, `\scriptfont2` и `\scriptscriptfont2` не содержит как минимум 22 параметра `\fontdimen`. Для краткости мы будем называть эти параметры от σ_1 до σ_{22}, где параметры берутся из `\textfont2`, если текущим является выделенный или текстовый стиль (D, D', T или T'), из `\scriptfont2` если текущим является стиль S или S', и из `\scriptscriptfont2` в оставшемся случае. Аналогично, каждый из трех шрифтов семейства 3 должен иметь как минимум 13 параметров `\fontdimen`, и мы обозначаем их от ξ_1 до ξ_{13}. Запись ξ_9, например, обозначает девятый параметр `\scriptfont3`, если TEX в это время печатает что-нибудь в `\scriptstyle`.

Математический список — это последовательность элементов различных типов, перечисленных в главе 17. TEX набирает формулу, преобразовывая ее математический список в горизонтальный. Когда начинается такой набор, TEXв дополнение к самому математическому списку имеет еще две порции информации. (a) Начальный стиль говорит, какой стиль должен использоваться для математического списка, если только элемент стиля не задает другой стиль. Например, начальный стиль для выделенных формул — это D, для уравнений в тексте или для номера уравнения — T, а для подформул это может быть любой из восьми стилей, определенных в главе 17. Для обозначения текущего стиля мы будем использовать C и будем говорить, что математический список печатается в стиле C. (b) Печать производится либо со штрафами, либо нет. Формулы в тексте абзаца преобразуются в горизонтальный список, в котором после бинарных операций и отношений вставляются дополнительные элементы штрафа для того, чтобы помочь при разбиении строк. Такие штрафы в других случаях не вставляются, поскольку от них не было бы никакой пользы.

Рассмотрим восемь стилей $D > D' > T > T' > S > S' > SS > SS'$, в порядке уменьшения. Так, $C \le S$ означает, что текущим является стиль S, S', SS или SS'. Стиль C' означает текущий стиль, к которому добавлен штрих, если его там еще нет. Например, мы имеем $C' = T'$ тогда и только тогда, когда $C = T$ или $C = T'$. Стиль $C\uparrow$ является верхним индексным стилем для C. Это означает стиль S, если C равно D или T, стиль S', еслиC равно D' или T', стиль SS, если C равно S или SS и стиль SS', если C равно S' или SS'. Наконец, стиль $C\downarrow$ — это стиль нижнего индекса, который равен $(C\uparrow)'$.

В главе 17 говорилось, что наиболее важные компоненты математических списков называются атомами и что каждый атом имеет три поля: ядро, верхний индекс и нижний индекс. Часто требуется выполнять такую подпрограм-

му: "Установите бокс x в такое-то такое-то поле в стиле таком-то таком-то". Это означает: (a) если указанное поле пусто, x устанавливается равным нулевому боксу; (b) если поле содержит символ, x устанавливается в h-бокс, содержащий этот символ в подходящем размере, и в ширину бокса включается курсивная поправка для символа; (c) если поле содержит математический или горизонтальный список, x устанавливается в h-бокс, содержащий результат печати этого списка в заданном начальном стиле. В случае (c) клей устанавливается не растяжимый и не сжимаемый, а дополнительный уровень h-боксирования, если он оказывается лишним, опускается.

Другая подпрограмма устанавливает бокс x в переменный ограничитель, имеющий указанный минимум высоты плюс глубины. Это означает, что поиск проводится следующим образом. Ограничитель определяется двумя символами, "маленьким символом" a в семействе f и "большим символом" b в семействе g. Сначала ищется символ a в шрифте повторного индекса f, если $C \le SS$, затем a в индексном шрифте f, если $C \le S$, затем a в текстовом шрифте f. Если не нашлось ничего подходящего среди a и f, то таким же образом проверяется альтернатива в b и g. Либо (a, f), либо (b, g) может оказаться равным $(0, 0)$, что означает, что соответствующая часть поиска может быть обойдена. Когда в шрифте рассматривается символ, поиск немедленно прекращается, если этот символ имеет удовлетворительную высоту плюс глубину, либо если он является расширяемым. Более того, если символ не прекращает поиск, но имеет последователя, следующим рассматривается последователь. (Для дальнейшей информации о последователях и расширяемых символах смотри руководство METAFONT или системную документацию по файлам `tfm`.) Если поиск доходит до конца и не находит подходящего символа, выбирается символ, имеющий максимальную высоту плюс глубину. Если вообще не было найдено никакого символа (либо потому что $a = f = b = g = 0$, либо потому что такого символа в шрифтах не существует), x устанавливается в пустой бокс, ширина которого равна `\nulldelimiterspace`. Если найден расширяемый символ, x устанавливается в вертикальный бокс, содержащий достаточно элементов, чтобы построить символ достаточного размера. Высотой этого бокса будет высота самого верхнего элемента, а шириной — ширина повторяемой части. Иначе x устанавливается в горизонтальный бокс, содержащий символ, который был найден. В ширину этого бокса включается курсивная поправка.

Существует также команда, которая "перебоксирует" данный бокс в бокс заданной ширины. Если бокс не имеет уже желательную ширину, TEX распаковывает его, (если только это был не вертикальный бокс), затем добавляется курсивная поправка, если она применялась, и вставляется клей `\hss` как слева, так и справа. Результирующий горизонтальный список упаковывается в h-бокс. Этот процесс используется, например, чтобы задать общую ширину для числителя и знаменателя дроби. Тот из них, что меньше, центрируется, если только в дополнение к добавляемому `\hss` нет бесконечного клея.

Если x — бокс, то для его высоты, глубины и ширины мы будем использовать, соответственно, аббревиатуры $h(x)$, $d(x)$ и $w(x)$.

Теперь приведем правила для набора данного математического списка в начальном стиле C. Обработка происходит слева направо, транслируя по очереди каждый элемент. Список обрабатывается за два прохода. Большая часть работы выполняется за первый проход, который накапливает конкретные преобразования математических элементов. Мы сначала рассмотрим эту часть задачи.

1. Если текущий элемент является линейкой, отпадающим элементом, штрафом, "whatsit" или граничным элементом, просто оставьте его неизменным и переходите к следующему элементу.

2. Если текущий элемент — это клей или керн, преобразуйте его следующим образом: если это клей из `\nonscript`, проверьте, не является ли следующий элемент клеем или керном, и если это так, то при $C \le S$ уберите этот элемент. Иначе, если текущий элемент из `\mskip` или `\mkern`, преобразуйте его из `mu` в абсолютные единицы умножением каждого конечного размера на $\frac{1}{18}\sigma_6$. Затем переходите к следующему элементу.

3. Если текущий элемент является изменением стиля, установите C в указанный стиль. Текущий стиль удалите из списка и переходите к следующему элементу.

4. Если текущий элемент является выбором одного из 4-х путей, он содержит четыре математических списка для четырех основных стилей. Замените его на математических список, соответствующий текущему стилю C, затем переходите к первому из необработанных элементов.

5. Если текущий элемент является атомом Bin и если он был первым атомом в списке или если самый последний из предыдущих атомов был Bin, Op, Rel, Open или Punct, измените текущий Bin на Ord и продолжайте с правила 14. Иначе продолжайте с правила 17.

6. Если текущий элемент является атомом Rel, Close или Punct и если самый последний из предшествующих атомов был Bin, замените этот предшествующий Bin на Ord. Продолжайте с правила 17.

7. Если текущий элемент — это атом Open или Inner, отправляйтесь прямо к правилу 17.

8. Если текущий элемент — это атом Vcent (из `\vcenter`), пусть его ядром является вертикальный бокс, высота плюс глубина которого равна v. Измените его высоту на $\frac{1}{2}v+a$, а глубину на $\frac{1}{2}v-a$, где a — это осевая высота σ_{22}. Измените этот атом на тип Ord и продолжайте с правила 17.

9. Если текущим элементом является атом Over (из `\overline`), установите бокс x в ядро в стиле C'. Затем замените ядро на v-бокс, содержащий ядро θ, горизонтальную линейку высотой θ, ядро 3θ, и бокс x, сверху вниз, где $\theta = \xi_8$ — это толщина линейки по умолчанию. (Это помещает линейку над ядром, причем над этой линекой очищается 3θ и добавляется θ единиц пробела.) Продолжайте с правила 16.

10. Если текущим элементом является атом Under (из `\underline`), установите бокс x в ядро в стиле C. Затем замените ядро на столбец, сделанный из бокса x, ядра 3θ и горизонтальной линейки высотой θ, где $\theta = \xi_8$ — это толщина линейки по умолчанию, и добавьте θ к глубине бокса. (Это помещает линейку под ядром, предполагая, что ниже линейки располагается 3θ очистки и θ единиц пробела.) Продолжайте с правила 16.

11. Если текущий элемент является атомом Rad, (из `\radical`, например, `\sqrt`), установите бокс x в ядро в стиле C'. Пусть $\theta = \xi_8$ и пусть $\varphi = \sigma_5$, если $C > T$, иначе $\varphi = \theta$. Установите $\psi = \theta + \frac{1}{4}|\varphi|$. Это минимум очистки, которая будет позволена между боксом x и линейкой, расположенной над ним. Установите

бокс y в переменный ограничитель для этого атома радикала, имеющий высоту плюс глубину $h(x)+d(x)+\psi+\theta$ или более. Затем установите $\theta \leftarrow h(y)$. Это толщина линейки, которая используется в конструкции радикала. (Заметим, что дизайнер шрифта указал толщину линейки, делая ее высотой символа радикала. Базовая линия символа должна быть в точности под линейкой.) Если $d(y) > h(x)+d(x)+\psi$, увеличьте ψ на половину излишка, т.е., установите $\psi \leftarrow \frac{1}{2}(\psi + d(y) - h(x) - d(x))$. Создайте v-бокс, состоящий из ядра θ, горизонтальной линейки высотой θ, ядра ψ, и бокса x, сверху вниз. Ядро атома радикала теперь заменено на бокс y, поднятый на $h(x) + \psi$, за которым следует новый v-бокс. Продолжайте с правила 16.

12. Если текущим элементом является атом Acc (из `\mathaccent`), то, если в текущем размере не существует символа акцента, просто переходите к правилу 16. Иначе установите бокс x в ядро в стиле C', а u в ширину этого бокса. Если ядро — это не простой символ, пусть $s = 0$. Иначе установите s в величину керна для ядер, за которыми следует `\skewchar` их шрифта. Если символ акцента имеет в своем шрифте последователя, ширина которого $\leq u$, замените его на последователя и повторите это предложение. Теперь пусть $\delta \leftarrow \min(h(x), \chi)$, где χ — это `\fontdimen5` (x-высота) в шрифте акцента. Если ядро является простым символом, замените бокс x на бокс, содержащий ядро вместе с верхним и нижним индексами атома Acc в стиле C и сделайте верхний/нижний индексы атома Acc пустыми. Также увеличьте δ на разность между новыми и старыми значениями $h(x)$. Поместите акцент в новый бокс y, включая курсивную поправку. Пусть z — это v-бокс, состоящий из бокса y, сдвинутого вправо на $s + \frac{1}{2}(u - w(y))$, керна $-\delta$ и бокса x. Если $h(z) < h(x)$, добавьте керн $h(x) - h(z)$ над боксом y и установите $h(z) \leftarrow h(x)$. Наконец, установите $w(z) \leftarrow w(x)$, замените ядро атома Acc на бокс z и продолжайте с правила 16.

13. Если текущий элемент является атомом типа Op, пометьте этот атом как имеющий пределы, если он был помечен при помощи `\limits` или был помечен `\displaylimits` и $C > T$. Если ядро не символ, установите $\delta \leftarrow 0$ и переходите к правилу 13a. Иначе, если $C > T$ и если символ ядра имеет в своем шрифте последователя, перейдите к последователю. (Это тот случай, когда операторы типа $\sum$ и $\int$ изменяются на больший размер в выделенных стилях.) Поместите символ в новый бокс x в текущем размере и установите δ в курсивную поправку для символа. Включайте δ в ширину бокса x тогда и только тогда, когда установлены пределы или нет нижнего индекса. Сдвиньте бокс x вниз на $\frac{1}{2}(h(x) + d(x)) - a$, где $a = \sigma_{22}$, так что символ оператора вертикально центрируется на оси. Этот сдвинутый бокс становится ядром атома Op.

13a. Если для этого атома Op не должны печататься пределы, переходите к правилу 17. Инач пределы присоединятся следующим образом: установите бокс x в поле верхнего индекса в стиле $C\uparrow$, установите бокс y в поле ядра в стиле C и установите бокс z в поле нижнего индекса в стиле $C\downarrow$. Перебоксируйте все три этих бокса в ширину $\max(w(x), w(y), w(z))$. Если поле верхнего индекса не пусто, присоедините бокс x над боксом y, отделив его керном размера $\max(\xi_9, \xi_{11} - d(x))$ и сдвинув бокс x вправо на $\frac{1}{2}\delta$. Поместите также керн величины ξ_{13} под боксом x. Если поле нижнего индекса не пусто, присоедините бокс z снизу бокса y, отделив его керном размера $\max(\xi_{10}, \xi_{12} - h(z))$ и сдвинув бокс z влево на $\frac{1}{2}\delta$. Поместите также керн размера ξ_{13} ниже бокса z. Результирующий v-бокс станет ядром текущего атома Op. Переходите к следующему элементу.

14. Если текущим элементом является атом Ord, переходите к правилу 17, если только не справедливо все следующее: ядро является символом, верхний и нижний индекс оба пусты, самый близкий символ в математическом списке — это атом типа Ord, Op, Bin, Rel, Open, Close или Punct и ядро следующего элемента является символом того же семейства, что и семейство настоящего атома Ord. В таких случаях настоящий символ помечается как текстовый символ. Если информация шрифта показывает лигатуру между этим и следующим символом, используя заданное семейство и текущий размер, то удалите настоящий атом, вставьте символ лигатуры в символ следующего элемента и передвиньтесь на этот элемент. Иначе, если информация шрифта указывает керн между текущим и следующим символом, вставьте керн после текущего атома Ord и после этого передвиньтесь на следующий элемент. Иначе (т.е., если между настоящим текстовым символом и последующим символом не указано ни лигатуры, ни керна), переходите к правилу 17.

15. Если текущий элемент является обобщенной дробью (и ему лучше ей быть, поскольку, если не применялись правила 1–14, осталась только эта возможность), пусть θ — это толщина черты и пусть (λ, ρ) — это левый и правый ограничители. Если эта дробь генерировалась при помощи `\over` или `\overwithdelims`, то $\theta = \xi_8$. Если она генерировалась при помощи `\atop` или `\atopwithdelims`, $\theta = 0$. Иначе она генерировалась при помощи `\above` или `\abovewithdelims`, и в этот момент было задано специфическое значение θ. Значение λ и ρ равны нулю, если только у нас не дробь с "ограничениями".

15a. Поместите числитель в бокс x, используя стиль T или T', если C равен D или D', иначе используя стиль $C{\uparrow}$. Поместите знаменатель в бокс z, используя стиль T', если $C > T$, иначе используя стиль $C{\downarrow}$. Если $w(x) < w(z)$, перебоксируйте x в ширину $w(z)$. Если $w(z) < w(x)$, перебоксируйте z в ширину $w(x)$.

15b. Если $C > T$, установите $u \leftarrow \sigma_8$ и $v \leftarrow \sigma_{11}$. Иначе установите $u \leftarrow \sigma_9$ или σ_{10}, в соответствии с тем, что $\theta \neq 0$ или $\theta = 0$, и установите $v \leftarrow \sigma_{12}$. (Дробь будет печататься так, что ее числитель будет сдвинут вверх на величину v относительно текущей базовой линии, а знаменатель сдвинут вниз на v, если только боксы не очень большие.)

15c. Если $\theta = 0$ (`\atop`), числитель и знаменатель комбинируются следующим образом. Установите $\varphi \leftarrow 7\xi_8$ или $3\xi_8$, в зависимости от того, $C > T$ или $C \leq T$; φ — это минимальная очистка, которая будет допускаться между числителем и знаменателем. Пусть $\psi = (u - d(x)) - (h(z) - v)$ будет реальной очисткой, которая получилась бы при текущих значениях u и v. Если $\psi < \varphi$, добавьте $\frac{1}{2}(\varphi - \psi)$ как к u, так и к v. Затем постройте v-бокс высотой $h(x) + u$ и глубиной $d(z) + v$, состоящий из бокса x, за которым следует подходящий керн, а за ним бокс z.

15d. Если $\theta \neq 0$ (`\over`), числитель и знаменатель комбинируются так. Установите $\varphi \leftarrow 3\theta$ или θ, в зависимости от $C > T$ или $C \leq T$; φ — это минимальное расстояние, которое допускается между числителем и знаменателем и чертой. Пусть $a = \sigma_{22}$ — это текущая осевая высота. Середина черты будет помещена на этой высоте. Если $(u - d(x)) - (a + \frac{1}{2}\theta) < \varphi$, увеличьте u на разность между этими величинами, а если $(a - \frac{1}{2}\theta) - (h(z) - v) < \varphi$, увеличьте на разность v. Наконец, постройте v-бокс высотой $h(x) + u$ и глубиной $d(z) + v$, состоящий из бокса x, за которым следует керн, за ним горизонтальная линейка высотой θ, за ней другой

керн, за ним бокс z, где керны вычислены так, что верх линейки оказывается на $a - \frac{1}{2}\theta$ выше базовой линии.

15e. Заключите v-бокс, который был построен по правилу 15c или 15d в ограничители, высота плюс глубина которых равна, как минимум, σ_{20}, если $C > T$, и как минимум σ_{21} в других случаях. Сдвиньте ограничители вверх или вниз, так чтобы они были вертикально центрированы относительно оси. Замените обобщенную дробь на атом Inner, ядром которого является результирующая последовательность трех боксов (левый ограничитель, v-бокс, правый ограничитель).

Правила 1–15 объясняют предварительную обработку элементов математического списка. Но мы еще не сказали, как печатаются верхние и нижние индексы. Поэтому некоторые из этих правил приводят к следующему пост-процессу:

16. Измените текущий элемент на атом Ord и продолжайте с правила 17.

17. Если ядро текущего элемента является математическим списком, замените его на бокс, который получается при наборе этого списка в текущем стиле. Затем, если ядро не простой символ, перейдите к правилу 18. Иначе мы находимся в общем случае, когда математический список переводится в эквивалентный ему горизонтальный список. Преобразуйте символ в символьный бокс для указанного семейства в текущем размере. Если символ не помечен как текстовый символ выше в правиле 14 или если параметр номер 2 `\fontdimen` его шрифта равен нулю, установите δ в курсивную поправку, иначе установите δ в нуль. Если δ не равна нулю и если поле нижнего индекса текущего атома пусто, вставьте после символьного бокса керн шириной δ и установите δ в нуль. Продолжайте с правила 18.

18. (Осталось для текущего атома присоединить возможные нижние и верхние индексы.) Если поля верхнего и нижнего индексов пусты, переходите к следующему элементу. Иначе продолжайте со следующих подправил.

18a. Если ядро переведено в символьный бокс с возможно следующим за ним керном, установите u и v равными нулю. Иначе установите $u \leftarrow h - q$ и $v \leftarrow d + r$, где h и d — высота и глубина приведенного ядра, а q и r — значения σ_{18} и σ_{19} в шрифте, соответствующем стилям $C{\uparrow}$ и $C{\downarrow}$. (Величины u и v представляют собой минимальные величины, на которые верхний и нижний индексы будут сдвигаться вверх и вниз. Эти предварительно определенные значения u и v могут быть позднее увеличены.)

18b. Если поле верхнего индекса пусто (так что есть только верхний индекс), установите бокс x в нижний индекс в стиле $C{\downarrow}$ и добавьте `\scriptspace` к $w(x)$. Присоедините этот бокс к трансляции текущего элемента, сдвигая его вниз на $\max(v, \sigma_{16}, h(x) - \frac{4}{5}|\sigma_5|)$, и переходите к следующему элементу. (Идея заключается в том, чтобы получить уверенность, что верхний индекс сдвинут как минимум на v и как минимум на σ_{16}. Более того, верхушка нижнего индекса не должна простираться выше $\frac{4}{5}$ текущей x-высоты.)

18c. Установите бокс x в поле верхнего индекса в стиле $C{\uparrow}$ и добавьте значение `\scriptspace` к $w(x)$. Затем установите $u \leftarrow \max(u, p, d(x) + \frac{1}{4}|\sigma_5|)$, где $p = \sigma_{13}$ если $C = D$, $p = \sigma_{15}$, если $C = C'$ и $p = \sigma_{14}$ в других случаях. Это дает пробное положение для верхнего индекса.

18d. Если пусто поле нижнего индекса (так что есть только верхний индекс), присоедините бокс x к трансляции текущего атома, сдвигая его вверх на u, и переходите к следующему элементу. Иначе (т.е., когда присутствует как нижний, так и верхний индесы), установите бокс y в нижний индекс в стиле $C{\downarrow}$, добавьте `\scriptspace` to $w(y)$ и установите $v \leftarrow \max(v, \sigma_{17})$.

18e. (Осталось расположить объединенную комбинацию нижнего/верхнего индекса.) Пусть $\theta = \xi_8$ — это толщина линейки по умолчанию. Если $(u - d(x)) - (h(y) - v) \geq 4\theta$, переходите к правилу 18f. (Это означает, что пробел между нижним и верхним индексом равен, как минимум, 4θ.) Иначе переставьте v так, что $(u - d(x)) - (h(y) - v) = 4\theta$. Пусть $\psi = \frac{4}{5}|\sigma_5| - (u - d(x))$. Если $\psi > 0$, увеличьте u на ψ и уменьшите v на ψ. (Это означает, что низ верхнего индекса будет над базовой линией на расстоянии, не меньшем $\frac{4}{5}$ x-высоты.)

18f. Наконец, пусть δ равно нулю, если только оно не было установлено в ненулевое значение правилами 13 или 17. (Это величина горизонтального смещения между нижним и верхним индексом.) Создайте v-бокс высотой $h(x)+u$ и глубиной $d(y) + v$, состоящий из бокса x, сдвинутого вправо на δ, за которым следует подходящий керн, а за ним бокс y. Присоедините этот v-бокс к трансляции текущего элемента и переходите к следующему.

После того, как по правилам 1–18 обработан весь математический список, TEX смотрит на последний атом (если он был) и изменяет его тип с Bin на Ord (если у него был тип Bin). Затем выполняется следующее правило:

19. Если математический список начинается и кончается граничными элементами, вычислите максимальную высоту h и глубину d боксов в трансляции математического списка, которая была сделана на первом проходе, принимая во внимание тот факт, что некоторые боксы могут быть подняты или опущены. Пусть $a = \sigma_{22}$ — это осевая высота и пусть $\delta = \max(h - a, d + a)$ — это величина, на которую расширяется формула за ось. Замените граничные элементты на ограничители, высота плюс глубина которых как минимум равна $\max(\lfloor \delta/500 \rfloor f, 2\delta - l)$, где f — это `\delimiterfactor`, а l — это `\delimitershortfall`. Сдвиньте ограничители вверх или вниз так, чтобы они были вертикально центрированы по отношению к оси. Замените левый граничный элемент на атом Open, а правый граничный элемент на атом Close. (На этом шаге все вычисления делаются с C, равным начальному стилю математического списка. Элементы стиля в середине списка не влияют на стиль правого граничного элемента.)

20. Правила 1–19 преобразуют математический список в последовательность элементов, в которой остались только атомы типов Ord, Op, Bin, Rel, Open, Close, Punct и Inner. После того, как завершено это преобразование, делается второй проход через весь список, заменяя при этом все атомы на те боксы и керны, в которые они транслировались. Более того, перед каждым атомом кроме первого, вставляются дополнительные междуэлементные пробелы, основанные на типе этого и предшествующего атома. Междуэлементные пробелы определяются тремя параметрами `\thinmuskip`, `\medmuskip` и `\thickmuskip`. Единицы `mu` преобразуются в абсолютные единицы, как в правиле 2 выше. Глава 18 содержит схему, которая определяет междуэлементные пробелы, некоторые из которых являются `\nonscript`, т.е., вставляются только в стилях $> S$. Список мог также содержать

элементы стиля, которые убираются при втором проходе. Они используются для изменения текущего стиля только при первом проходе, так что оба прохода, когда они обрабатывают каждый конкретный атом, имеют одно и то же значение C.

21. Если математический список был частью абзаца, то кроме междуэлементных пробелов после трансляции каждого атома типа Bin или Rel помещаются штрафы. Штраф после Bin равен `\binoppenalty`, а штраф после Rel — `\relpenalty`. Однако, штраф не вставляется после последнего элемента списка, если его числовое значение ≥ 10000, если ближайший элемент в списке уже является штрафом или после атома Rel, за которым сразу же следует другой атом Rel.

22. После того, как выполнены предыдущие действия, математический список полностью преобразован в горизонтальный. Если результат вставляется в более крупный горизонтальный список в горизонтальной или частной горизонтальной моде, он заключается в элементы "включение математики" и "выключение математики", каждый из которых записывает текущее значение `\mathsurround`. Или, если этот список выделенная формула, он обрабатывается далее, как объяснялось в главе 19.

Обзор использования параметров. Приведем обещанный указатель, который отсылает ко всему, что касается таинственных параметров в символьных шрифтах. Если вы внимательно изучите правила, это позволит вам получать самые лучшие результаты, устанавливая подходящие значения параметров для новых шрифтов, которые можно использовать в математической печати. У каждого шрифтового параметра есть внешнее имя, которое применяется в поддерживающем пакете математического обеспечения. Например, на σ_{14} обычно ссылаются как на "sup2", ξ_8 — это "толщина_линейки_по умолчанию'. Эти внешние имена указаны в таблице.

Параметр		*Использован в*	*Параметр*		*Использован в*
σ_2	space	17	σ_{17}	sub2	18d
σ_5	x_height	11, 18b, 18c, 18e	σ_{18}	sup_drop	18a
σ_6	quad	2, 20	σ_{19}	sub_drop	18a
σ_8	num1	15b	σ_{20}	delim1	15e
σ_9	num2	15b	σ_{21}	delim2	15e
σ_{10}	num3	15b	σ_{22}	axis_height	8, 13, 15d, 19
σ_{11}	denom1	15b	ξ_8	default_rule_thickness	9, 10, 11, 15, 15c, 18e
σ_{12}	denom2	15b	ξ_9	big_op_spacing1	13a
σ_{13}	sup1	18c	ξ_{10}	big_op_spacing2	13a
σ_{14}	sup2	18c	ξ_{11}	big_op_spacing3	13a
σ_{15}	sup3	18c	ξ_{12}	big_op_spacing4	13a
σ_{16}	sub1	18b	ξ_{13}	big_op_spacing5	13a

Кроме символьных и расширенных шрифтов (семейства 2 и 3), приведенные выше правила также ссылаются на параметры других семейств. Правило 17 использует параметр 2 `\fontdimen` (пробел), чтобы определить, надо ли вставлять курсивную поправку между соседними буквами, а правило 12 использует параметр 5 (x_высоту), для расположения символа акцента. На набор математики также влияют некоторые шрифтовые параметры: параметры размера `delimiterlimit`

(правило 19), `\nulldelimiterspace` (в конструкции переменных ограничителей для правил 11, 15e, 19), `\mathsurround` (правило 22) и `\scriptspace` (правило 18bcd); целые параметры `\delimiterfactor` (правило 19), `\binoppenalty` (правило 21) и `\relpenalty` (правило 21); параметры математического клея `\thinmuskip`, `\medmuskip` и `\thickmuskip` (правило 20).

Горе тому автору, который всегда хочет поучать!
Чтобы быть скучным, есть один секрет: договаривайте все до конца.

Woe to the author who always wants to teach!
The secret of being a bore is to tell everything.

— VOLTAIRE, *De la Nature de l'Homme* (1737)

Very few Compositors are fond of Algebra, and rather chuse to be employed upon plain work.

— PHILIP LUCKOMBE, *The History and Art of Printing* (1770)

Н

Перенос

Иногда лучше разорвать слово дефисом, чем слишком сильно растягивать промежутки между словами. Поэтому TEX, когда нет хорошей альтернативы, пытается поделить слова на слоги.

Но компьютеры издавна славятся плохими переносами. Когда был полностью автоматизирован набор газет, вскоре начали ходить шутки о "therapists who pre-ached on wee-knights".

Не трудно понять, почему компьютерам трудно справиться с такой задачей, поскольку проблема переноса вообще очень сложна. Например, существительное "record" полагается переносить как "rec-ord", а глагол "record" — как "re-cord". Само слово "hyphenation"(перенос) — это что-то исключительное: если "hy-phen-a-tion" сравнить с похожим на него словом "con-cat-e-na-tion", нет никакой ясности, почему буква n в одном случае должна быть присоединена к букве e, а в другом — нет. Сравнение таких слов, как "dem-on-stra-tion" и "de-mon-stra-tive" показывает, что замена двух букв может повлиять на положение дефиса на расстоянии девяти позиций от них.

В 1980–1982 году Фрэнком М. Лайэнгом было предложено хорошее решение проблемы переноса, которое и применяет TEX. Алгоритм Лайэнга работает быстро и находит почти все позиции, в которые разрешено вставлять дефис. Кроме того, он почти не делает ошибок и занимает сравнительно мало памяти. Сверх того, этот метод достаточно гибкий и может быть адаптирован к любому языку, а также может использоваться для переноса слов на двух языках одновременно. Тезисы Ph.D. Лайэнга, опубликованные в 1983 году Компьютерным отделением Стэнфордского университета, объясняют, как взять словарь переносов и обучить ему TEX, то есть объясняет, как рассчитать таблицы, по которым TEX без ошибок сможет воспроизвести большинство переносов этого словаря.

Когда TEX переносит слово, он сначала ищет его в "словаре исключений", который указывает положения дефисов для слов, требующих специальной обработки. Если в словаре нет такого слова, TEX ищет в слове *образец*. Это и есть ключевая идея метода Лайэнга. Опишем, как работает этот метод, используя в качестве примера слово "hyphenation" в том случае, когда TEX оперирует английскими образцами формата начального TEX'а. Данное слово сначала с двух концов дополняется специальными маркерами, и мы получаем

```
.hyphenation.
```

где "." обозначает специальный маркер. Дополненное слово имеет подслова

```
. h y p h e n a t i o n .
```

длиной, равной одному,

```
.h hy yp ph he en na at ti io on n.
```

длиной, равной двум,

```
.hy hyp yph phe hen ena nat ati tio ion on.
```

длиной, равной трем, и так далее. Каждое подслово длины k является образцом, который определяет $k+1$ маленьких целых значений, связанных с желательностью дефисов в позициях между ним и соседними с ним буквами. Мы можем показать эти значения, присоединяя их в качестве индексов. Например, "$_0$h$_0$e$_2$n$_0$" означает, что значения, соответствующие подслову "hen" — это 0, 0, 2 и 0, где 2 относится к дефису между буквами e и n. Все межбуквенные значения равны нулю за исключением их значений для тех подслов, которые являются элементами в текущем TEXовском "словаре образцов". В данном случае специальными образцами являются только подслова

$$_0\mathtt{h}_0\mathtt{y}_3\mathtt{p}_0\mathtt{h}_0\ \ _0\mathtt{h}_0\mathtt{e}_2\mathtt{n}_0\ \ _0\mathtt{h}_0\mathtt{e}_0\mathtt{n}_0\mathtt{a}_4$$
$$_0\mathtt{h}_0\mathtt{e}_0\mathtt{n}_5\mathtt{a}_0\mathtt{t}_0\ \ _1\mathtt{n}_0\mathtt{a}_0\ \ _0\mathtt{n}_2\mathtt{a}_0\mathtt{t}_0\ \ _1\mathtt{t}_0\mathtt{i}_0\mathtt{o}_0\ \ _2\mathtt{i}_0\mathtt{o}_0$$

Далее TEX вычисляет максимальную межбуквенную величину, встречающуюся в каждом подслове, которая относится к каждой межбуквенной позиции. Например, к позиции между e и n в нашем случае относится четыре значения (2 из $_0$h$_0$e$_2$n$_0$, 0 из $_0$h$_0$e$_0$n$_0$a$_4$, 0 из $_0$h$_0$e$_0$n$_5$a$_0$t$_0$ и 1 из $_1$n$_0$a$_0$). Максимальным из них является 2. Результатом всех максимизаций будет

$$.\,_0\mathtt{h}_0\mathtt{y}_3\mathtt{p}_0\mathtt{h}_0\mathtt{e}_2\mathtt{n}_5\mathtt{a}_4\mathtt{t}_2\mathtt{i}_0\mathtt{o}_0\mathtt{n}_0.$$

Теперь переходим к последнему шагу. Дефис между буквами считается допустимым, если соответствующее межбуквенное значение *нечетное*. Так, найдены две потенциальные точки разрыва: "hy-phen-ation". Аналогично, слово "concatenation" содержит образцы

$$_0\mathtt{o}_2\mathtt{n}_0\ \ _0\mathtt{o}_0\mathtt{n}_1\mathtt{c}_0\ \ _1\mathtt{c}_0\mathtt{a}_0\ \ _1\mathtt{n}_0\mathtt{a}_0\ \ _0\mathtt{n}_2\mathtt{a}_0\mathtt{t}_0\ \ _1\mathtt{t}_0\mathtt{i}_0\mathtt{o}_0\ \ _2\mathtt{i}_0\mathtt{o}_0$$

и это дает "$_0$c$_0$o$_2$n$_1$c$_0$a$_0$t$_0$e$_1$n$_2$a$_1$t$_2$i$_0$o$_0$n$_0$", т.е., "con-cate-na-tion".

Давайте попробуем подобрать образцы начального TEX'а для 34-буквенного слова "supercalifragilisticexpialidocious". Они будут такими:

$$\mathtt{u}_1\mathtt{pe}\ \mathtt{r}_1\mathtt{c}\ _1\mathtt{ca}\ \mathtt{al}_1\mathtt{i}\ \mathtt{ag}_1\mathtt{i}\ \mathtt{gil}_4\ \mathtt{il}_1\mathtt{i}\ \mathtt{il}_4\mathtt{ist}\ \mathtt{is}_1\mathtt{ti}\ \mathtt{st}_2\mathtt{i}$$
$$\mathtt{s}_1\mathtt{tic}\ _1\mathtt{exp}\ \mathtt{x}_3\mathtt{p}\ \mathtt{pi}_3\mathtt{a}\ _2\mathtt{i}_1\mathtt{a}\ \mathtt{i}_2\mathtt{al}\ _2\mathtt{id}\ _1\mathtt{do}\ _1\mathtt{ci}\ _2\mathtt{io}\ _2\mathtt{us}$$

(где нулевые индексы не показаны), а это дает

$$.\,_0\mathtt{s}_0\mathtt{u}_1\mathtt{p}_0\mathtt{e}_0\mathtt{r}_1\mathtt{c}_0\mathtt{a}_0\mathtt{l}_1\mathtt{i}_0\mathtt{f}_0\mathtt{r}_0\mathtt{a}_0\mathtt{g}_1\mathtt{i}_0\mathtt{l}_4\mathtt{i}_0\mathtt{s}_1\mathtt{t}_2\mathtt{i}_0\mathtt{c}_1\mathtt{e}_0\mathtt{x}_3\mathtt{p}_2\mathtt{i}_3\mathtt{a}_0\mathtt{l}_2\mathtt{i}_1\mathtt{d}_0\mathtt{o}_1\mathtt{c}_2\mathtt{i}_0\mathtt{o}_2\mathtt{u}_0\mathtt{s}_0.$$

Полученные переносы "su-per-cal-ifrag-ilis-tic-ex-pi-ali-do-cious" согласуются с Random House's *Unabridged Dictionary* (который к тому же показывает несколько большее:

"su-per-cal-i-frag-i-lis-tic-ex-pi-al-i-do-cious").

Начальный TEX загружает в память TEX'а в точности 4447 образцов, начиная с "$_0$.$_0$a$_0$c$_0$h$_4$" и кончая "$_4$z$_1$z$_2$" и "$_0$z$_4$z$_0$y$_0$". В этих образцах все межбуквенные значения находятся между 0 и 5. Максимальное нечетное значение, такое как 5 в "$_0$h$_5$e$_0$l$_0$o$_0$", приводит к желательной точке переноса в словах типа "bach-e-lor" и "ech-e-lon", в то время как большое четное значение типа 4 в "$_0$h$_0$a$_0$c$_0$h$_4$" подавляет нежелательный перенос в словах типа "tooth-aches". Лайэнг получил эти образцы, разбирая специальную версию карманного словаря *Webster's Pocket Dictionary* (Merriam, 1966), который содержит около 50 000 слов, включая производные формы. Затем он расположил предварительный набор образцов, полученный из этих данных, напротив современного словаря переносов из приблизительно 115 000 слов, полученного от издателя. Найденные при этом ошибки привели к добавлению к списку карманного словаря еще около 1000 слов, таких как camp-fire, Af-ghan-i-stan и bio-rhythm. Он тщательно взвесил наиболее предпочтительные переносы нескольких тысяч общеупотребительных слов, в результате образцы начального TEX'а гарантируют правильный перенос 700 или около того наиболее общеупотребительных английских слов, а так же общеупотребительных технических слов типа al-go-rithm. Эти образцы находят 89.3% переносов словаря Лайэнга, и не вставляют переносов, которых там нет.

Образцы, полученные из общепринятых слов языка, неплохо работают на необычных или заново образованных словах, которых нет в исходном словаре. Например, образцы Лайенга находят правильные подмножества для переноса в слове, которое все современные полные словари считают самым длинным в английском языке, а именно,

```
pneu-monoul-tra-mi-cro-scop-ic-sil-i-co-vol-canoco-nio-sis.
```

Они даже достаточно успешно справляются со словами из других языков, которые не слишком далеки от английского. Например, псевдонемецкое выражение из Mark Twain's *Connecticut Yankee* получается только с шестью или семью плохими переносами:

```
Con-stanti-nop-o-li-tanis-cher-
  dudel-sack-spfeifen-mach-ers-ge-sellschafft;
Ni-hilis-ten-dy-na-mitthe-
  aterkaestchensspren-gungsat-ten-taetsver-suchun-gen;
Transvaal-trup-pen-tropen-trans-port-
  tram-pelth-iertreib-er-trau-ungsthrae-nen-tra-goedie;
Mekka-musel-man-nen-massen-menchen-
  mo-er-der-mohren-mut-ter-mar-mor-mon-u-menten-machen.
```

Но когда начальный TEX пытается справиться с именем известного Welsh города

```
Llan-fair-p-wll-gwyn-gyll-gogerych-
  wyrn-drob-wl-l-l-lan-tysil-i-o-gogogoch,
```

становятся очевидными лингвистические различия, поскольку правильными будут такие переносы:

```
Llan-fair-pwll-gwyn-gyll-go-ger-y-
  chwyrn-dro-bwll-llan-ty-sil-i-o-go-go-goch.
```

Для других языков можно получить подходящие значения образцов, применяя метод Лайенга к подходящим словарям переносов.

Точки зрения различных английских словарей по поводу слогов не всегда совпадают. Например, словарь *American Heritage Dictionary* говорит "`in-de-pend-ent`", в то время как *Webster* указывает на "`in-de-pen-dent`". Начальный TEX обычно следует Webster'у, за исключением нескольких случаев, в которых он предпочитает другие алгоритмы.

[С этого места и до конца этого приложения TEX будет делать набор с

```
\hyphenpenalty=-1000   \pretolerance=-1   \tolerance=1000
\doublehyphendemerits=-100000  \finalhyphendemerits=-100000
```

так что переносы будут встречаться намного чаще, чем обычно.]

Тот факт, что начальный TEX находит только 90% разрешенных в большом словаре переносов, это, конечно, не повод для тревоги. Когда принимается во внимание частота появления слов, вероятность повышается сильно за 95%. Поскольку алгоритм TEX'а по разбиению строк часто находит хороший способ разбиения абзаца на строки, вообще не пользуясь переносом, и поскольку есть хорошие шансы найти точки переноса поблизости от той, которая пропущена образцами TEX'а, становится ясно, что ручное вмешательство для исправления или вставки переносов в результат работы TEX'а требуется редко и что такая отделка требует незначительного времени по сравнению с обычной работой по вводу и корректированию.

Но если вы обнаружите, что образцы не совсем правильно работают для вашей эадачи, то всегда можете вставить дополнительные слова в словарь исключений TEX'а. Например, английский оригинал этой книги печатался с добавлением трех исключительных слов: формат приложения E включал команду

```
\hyphenation{man-u-script man-u-scripts ap-pen-dix}
```

которая говорила TEX'у, как переносить слова "manuscript", "manuscripts" и "appendix". Заметим, что были введены как единственная, так и множественная форма слова "manuscript", поскольку словарь исключений дает переносы, только если слово полностью совпадает с элементом словаря. (Точные правила для команды `\hyphenation` обсуждаются ниже.)

Чтобы увидеть все переносы, которые начальный TEX будет находить в некотором тексте, можно сказать `\showhyphens{`⟨текст⟩`}`, и резуль-

тат появится на терминале (и в протокольном файле). Например,

```
*\showhyphens{random manuscript manuscripts appendix}
Underfull \hbox (badness 10000) detected at line 0
[] \tenrm ran-dom manuscript manuscripts ap-pendix
```

показывает позиции переноса, которые были найдены в этой книге, если не добавляется никаких исключений. Слово "manuscript" как-то проскользнуло через все обычные образцы. Для этой конкретной работы автор добавил его в качестве исключения, поскольку оно было использовано в книге 80 раз (не считая приложений).

Макрокоманда `\showhyphens` создает h-бокс, который умышленно недозаполнен, так что вы должны игнорировать предупреждающее сообщение о `badness 10000`. Это фальшивое предупреждение получается потому, что TEX выделяет переносы в компактной форме, только когда показывает содержимое ненормальных h-боксов. (Кудесники TEX'а могут получить удовольствие, изучая, как макрокоманда `\showhyphens` определена в приложении B.)

Если вы хотите добавить переносы одного или нескольких слов, просто скажите `\hyphenation{`⟨слова⟩`}`, где ⟨слова⟩ состоит из одного или нескольких элементов ⟨слово⟩, разделенных пробелами. ⟨Слово⟩ должно состоять из букв и дефисов. Более точно, "дефис" в этом контексте — это знак $\texttt{-}_{12}$, "буква" — это символьный элемент, номер категории которой равен 11 или 12, или команда, определенная при помощи `\chardef`, или `\char`⟨8-битное число⟩, так что соответствующий символ имеет ненулевой `\lccode`. TEX использует `\lccode`, чтобы преобразовать каждую букву в "строчную" форму. Переносимое слово находит пару в словаре исключений тогда и только тогда, когда оба слова после преобразования имеют одну и ту же строчную форму.

Как только TEX попытается переносить слово, которое совпадает с элементом словаря исключений, он будет вставлять возможные дефисы в указанных позициях, за исключением того, что дефисы никогда не вставляются после самой первой буквы и перед последней или предпоследней буквами слова. Чтобы позволить перенос в какой-нибудь позиции, надо вставить свой собственный дефис. Элемент `\hyphenation` мог вообще не содержать дефисов. Тогда TEX не будет вставлять в слово никаких дефисов.

Словарь исключений является глобальным, т.е., исключения в конце группы не исчезают. Если для одного и того же слова указано более одного `\hyphenation`, используется положение самых последних дефисов.

Словарь исключений является динамичным, а словарь образцов — статичным. Чтобы изменить текущий набор образцов переноса TEX'а, надо дать целиком новый набор, и TEX потратит некоторое время, чтобы привести его в вид, который делает эффективным алгоритм переноса. Формат команды `\patterns{`⟨образцы⟩`}`, где ⟨образцы⟩ — это последовательность элементов ⟨образец⟩, разделенных пробелами. Эта команда доступна только в `INITEX`,

а не в производственных версиях TEX'а, поскольку процесс сжатия образцов требует добавочной памяти, которая в производственных системах может быть использована лучше. `INITEX` посылает образцы и результат в форматный файл, который может с высокой скоростью загружаться производственными версиями.

В списке `\patterns` ⟨образец⟩ имеет более ограниченную форму, чем ⟨слово⟩ в списке `\hyphenation`, поскольку предполагается, что образцы готовятся экспертами, экспертиза которых хорошо оплачена. Каждый ⟨образец⟩ состоит из одной или нескольких ⟨значение⟩⟨буква⟩, за которыми следует ⟨буква⟩. Здесь ⟨значение⟩ либо пусто, либо является цифрой (от 0_{12} до 9_{12}). Пустое ⟨значение⟩ обозначает нуль. Например, образец "$_0a_1b_0$" может быть представлен как `0a1b0`, `a1b0`, `0a1b` или просто `a1b`. ⟨Буква⟩ — это символьный знак категории 11 или 12, `\lccode` которого не равен нулю. Чтобы в качестве ⟨буквы⟩ использовать цифру, вы должны поставить перед ней ненулевое ⟨значение⟩. Например, если по какой-нибудь причине нужен образец "$_1a_0 1_2$", можно получить его, напечатав `1a012`, в предположении, что `\lccode‘1` не равен нулю. Исключение: когда в образце появляется символ ".", он трактуется, как если бы его `\lccode` был равен 128. Код 128 (который не может быть реальным `\lccode`) используется TEX'ом, чтобы представить левую и правую границу слова, которое переносится.

Начальный TEX вводит файл, называемый `hyphen.tex`, который задает словарь образцов и первоначальный словарь исключений. Файл имеет вид

```
\patterns{.ach4 .ad4der .af1t .al3t  ···  zte4 4z1z2 z4zy}
\hyphenation{as-so-ciate as-so-ciates dec-li-na-tion oblig-a-tory
  phil-an-thropic present presents project projects reci-procity
  re-cog-ni-zance ref-or-ma-tion ret-ri-bu-tion ta-ble}
```

Первые тринадцать исключений предохраняют TEX от вставки неправильных переносов. Например, "`pro-ject`" и "`pre-sent`" — это слова типа "`re-cord`", которые нельзя переносить, не зная контекста. Другое исключение, "`ta-ble`", включено только чтобы удовлетворить условию, что начальный TEX полностью переносит 700 или около того наиболее употребительных слов английского языка.

Но каким образом TEX решает, что последовательность букв — это "слова", которые должны быть перенесены? Давайте вспомним, что TEX обрабатывает горизонтальный список, который в дополнение к простым символам содержит боксы, клей, линейки, лигатуры, керны, сбрасываемые элементы, метки, whatsits и так далее. Когда TEX не может найти подходящую точку разрыва без переноса слов, он как-то должен выбрать то, что он будет переносить. Слово не должно становиться неузнаваемым или непереносимым, если перед и/или после него присутствуют знаки препинания или если оно содержит лигатуры и керны. С другой стороны, желательно, чтобы перенос делался быстро, не тратя слишком много времени на разбор необычных ситуаций, в которых можно сделать перенос, но это трудно распознать механически.


TEX просматривает потенциально переносимые слова, начиная поиск с каждого клея, который не в математической формуле. Поиск обходит символы, `\lccode` которых равен нулю и лигатуры, которые начинаются с таких символов. Он также обходит whatsits и *неявные керны*, т.е., керны, которые вставлены самим TEX'ом из-за информации, хранящейся со шрифтом. Если поиск находит символ с ненулевым `\lccode` или лигатуру, которая начинается с такого символа, этот символ называется *начальной буквой*. Но если перед тем, как найдена подходящая начальная буква, встретится элемент любого другого типа, происходит отказ от переноса (до следующего клея). Таким образом, бокс, линейка или керн, явно вставленный при помощи `\kern` или `\/`, не должен вмешиваться между клеем и переносимым словом. Если начальная буква не строчная (т.е., если она не равна своему собственному `\lccode`), происходит отказ от переноса, если только `\uchyph` не положителен.


Если найдена подходящая начальная буква, пусть она принадлежит шрифту f. Если `\hyphenchar` f не между 0 и 255, происходит отказ от переноса. Если этот тест прошел, TEX продолжает сканирование, пока не наткнется на что-нибудь, что не является одним из трех "допустимых элементов": (1) символ шрифта f, чей `\lccode` не равен нулю; (2) лигатура, образованная целиком из символов типа (1); (3) неявный керн. Первый же недопустимый элемент прерывает эту часть процесса. Испытываемое слово состоит из всех букв, найденных в допустимых элементах. Заметим, что все эти буквы из шрифта f.


Если этим процессом найдено испытываемое слово $l_1 \ldots l_n$, переноса все еще не будет, если только $n > 4$. Более того, элементы, которые непосредственно следуют за испытываемым словом, должны состоять из нуля или более символов, лигатур и неявных кернов, за которыми непосредственно следуют клей, явный керн, штраф, whatsit, либо вертикальный материал из `\mark`, `\insert` или `\vadjust`. Таким образом, бокс, линейка, математическая формула или сбрасываемый элемент, находящиеся слишком близко к испытываемому слову, будут препятствовать переносу. (Поскольку TEX после явных дефисов вставляет пустой сбрасываемый элемент, из этих правил следует, что составное слово, уже имеющие дефисы, больше переноситься не будет).


Те испытываемые слова $l_1 \ldots l_n$, которые прошли все эти тесты, подчиняются описанному ранее алгоритму переноса. Дефисы не вставляются перед l_2 и после l_{n-2}. Если найдены другие точки переноса, в слово втавляются один или несколько сбрасываемых элементов. В это же время восстанавливаются лигатуры и неявные керны.


Поскольку лигатуры и керны рассматриваются совершенно одинаково, существует возможность, что одна точка переноса может мешать другой, поскольку лигатура с переносом может перекрываться лигатурой без переноса. Эта ненормальность в реальных жизненных ситуациях вероятно никогда не встретится, поэтому здесь не рассматривается интересный подход TEX'а к этой проблеме.


В соответствии с приведенными выше правилами есть важное различие между неявными и явными кернами, поскольку TEX, когда нахо-

дит как минимум одну точку переноса в слове, перевычисляет неявные керны. Если вы посмотрите внимательно, то сможете увидеть различие между этими двумя типами кернов, когда TEX показывает список элементов во внутреннем формате: "`\kern2.0`" обозначает неявный керн в 2 pt, а "`\kern 2.0`" — явный керн той же величины. Команда курсивной поправки `\/` вставляет явный керн.

Хотя решены еще не все проблемы переноса, тем не менее есть некоторый прогресс с того вечера, когда, согласно легенде, менеджеру фирмы RCA позвонил возмущенный клиент. Его компьютер RCA-301 только что перенес "God."

If all problems of hyphenation have not been solved, at least some progress has been made since that night, when according to legend, an RCA Marketing Manager received a phone call from a disturbed customer. His 301 had just hyphenated "God."

— PAUL E. JUSTUS, *There's More to Typesetting Than Setting Type* (1972)

The committee skeptically recommended more study for a bill to require warning labels on records with subliminal messages recorded backward.

— THE PENINSULA TIMES TRIBUNE (April 28, 1982)