С. М. Львовский

Набор и вёрстка в системе LaTeX

3-е издание, исправленное и
дополненное

2003

До 29 июля 2003 года разрешается некоммерческое использование книги С.М. Львовского "Набор и верстка в системе $\LaTeX$", размещенной на этом сайте, в неизменном виде и с сохранением копирайта, принадлежащего автору (в частности, эти материалы можно просматривать, печатать и копировать при условии, что распространение бесплатно, изменений не вносится, и к распространяемым материалам прилагается это уведомление).

Коммерческое использование без дополнительного согласования разрешается начиная с 29 июля 2003 года (в неизменном виде и с сохранением копирайта).

29 января 2003 г. С. Львовский

Оглавление

Предисловие

...а латекс — это такой редактор под линухом.
Из истории с сайта `www.anekdot.ru`

В реплике из эпиграфа все неверно: LaTeX не является текстовым редактором, работает отнюдь не только под операционной системой Linux (хотя и под ней тоже), наконец, его название произносится не «латекс», а «латех». Так что же такое LaTeX?

Если отвечать одной фразой, это издательская система на базе TeX'а.

Система компьютерной верстки TeX (произносится «тех») была создана выдающимся американским математиком и программистом Дональдом Кнутом в конце 70-х годов XX века; издательские системы на ее базе по сию пору широко используются и сдавать позиции не собираются. Чем объясняется столь редкое в компьютерном мире долголетие?

На первый взгляд, все свидетельствует против TeX'а. В самом деле, в отличие, допустим, от популярного ныне Microsoft Word'а, TeX не является системой типа WYSIWYG (What You See Is What You Get): чтобы посмотреть, как будет выглядеть на печати набираемый текст, надо запустить отдельную программу. И по структуре файлов TeX несовместим с Word'ом (что не удивительно: эта структура мало изменилась с начала 1980-х годов, когда никакого Word'а не было и в помине, а форматы Word-файлов меняются чуть ли не ежегодно). Наконец, чтобы работать в TeX'е, надо потратить определенное время на его изучение: трудно представить себе книгу под названием «TeX for dummies» («TeX для болванов»).

И что же в этом TeX'е хорошего? — спросит читатель. Вот краткий перечень TeX'овских достоинств:

1) Никакая другая из существующих в настоящее время издательских систем не может сравниться с TeX'ом в полиграфическом качестве текстов с математическими формулами.

2) Система TeX реализована на всех современных компьютерных платформах, и все эти реализации действительно работают одинаково.

3) Благодаря этому TeX стал международным языком для обмена математическими и физическими статьями: набрав свою статью в TeX'е, математик может послать ее по электронной почте своему коллеге, даже если отправитель работает под Windows, а получатель — с UNIX'ом или, допустим, на Макинтоше.

4) В Интернете существуют обширные «архивы препринтов», в которые каждый может послать (и из которых каждый может получить) статью; все эти статьи набраны опять-таки в TeX'е.

5) Наконец, основные реализации TeX'а для всех платформ распространяются бесплатно.

Разумеется, у TeX'а есть и недостатки. Главный из них — в том, что с помощью TeX'а тяжело (хотя в принципе и возможно) готовить тексты со сложным расположением материала на странице (наподобие рекламных буклетов). Для таких приложений, практически не встречающихся в научно-технической литературе, TeX не предназначен.

Настоящее пособие посвящено популярной издательской системе, основанной на TeX'е, — системе LaTeX. Оно пригодится как читателю, которому необходимо по роду своей работы готовить тексты с формулами, так и специалисту по компьютерной верстке. Для чтения значительной части книги никаких познаний в программировании или полиграфии не требуется: достаточен минимальный опыт работы на персональном компьютере в качестве пользователя. Более сложные разделы, сосредоточенные по большей части в конце книги, предназначены в первую очередь для полиграфистов.

Книга подготовлена, как легко догадаться, с помощью системы LaTeX (с использованием пакета XY-pic); исходные файлы книги являются свободно распространяемыми и сейчас размещаются по адресу `ftp://ftp.mccme.ru/pub/tex/llang.zip` (там же есть PostScript- и PDF-файлы).

Выражаю глубокую благодарность А. Шеню, без многочисленных бесед и споров с которым эта книга никогда не была бы написана (собственно говоря, она возникла из попытки дополнить книгу [4], представляющую собой пересказ с немецкого языка пособия [3] с дополнениями). Я благодарен И. А. Маховой за редактирование второго издания этой книги. При подготовке настоящего издания мне очень помогли беседы с М. Ю. Пановым, В. Ю. Радионовым и А. А. Черепановым; приложения написаны совместно с А. Шенем и В. В. Шуваловым; В. М. Гуровиц и В. В. Кондратьев нашли в тексте множество замечательных ошибок и опечаток. Last but not least, я рад выразить глубокую признательность В. Д. Арнольду за большую дружескую поддержку во время работы над всеми изданиями этого пособия.

С. Львовский

Глава I

Элементарное введение

1. Общие замечания

1.1. Что такое TeX и LaTeX

Как уже отмечалось в предисловии, TeX (произносится «тех», пишется также «TeX») — это созданная американским математиком и программистом Дональдом Кнутом (Donald E. Knuth) система для верстки текстов с формулами. Сам по себе TeX представляет собой специализированный язык программирования (Кнут не только придумал язык, но и написал для него транслятор, причем таким образом, что он работает совершенно одинаково на самых разных компьютерах), на котором пишутся издательские системы, используемые на практике. Точнее говоря, каждая издательская система на базе TeX'а представляет собой пакет макроопределений (макропакет) этого языка. В частности, LaTeX (произносится «латех» или «лэйтех», пишется также «LaTeX») — это созданная Лесли Лэмпортом (Leslie Lamport) издательская система на базе TeX'а.

Прежде чем углубиться в изучение собственно LaTeX'а, скажем несколько слов о других издательских системах на базе TeX'а. Наряду с LaTeX'ом распространены также макропакеты Plain TeX и AMS-TeX. Макропакет Plain TeX был разработан самим Дональдом Кнутом, рассматривавшим его в качестве платформы для построения более сложных систем; на практике он используется и как средство для обмена текстами (текст, подготовленный для Plain TeX'а, сравнительно несложно переделать в исходный текст для того же LaTeX'а). Что касается AMS-TeX'а, то эта издательская система сориентирована на важный, но узкий круг приложений: верстку статей для математических журналов и книг, издаваемых Американским математическим обществом. Соответственно, в AMS-TeX'е предусмотрено большое количество весьма изощренных возможностей для создания сложных математических формул, но при этом нет многих вещей, которые естественно было бы ожидать в издательских системах общего назначения (например, автоматической нумерации частей документа). Современные версии LaTeX'а включают в себя $\mathcal{AMS}$-TeX'овские возможности для набора формул (и используются Американским математическим обществом).

Настоящее издание этой книги посвящено описанию LaTeX'а версии 2_ε (она называется LaTeX 2_ε, произносится «два е» или «два эпсилон»). Эта версия, вышедшая в 1994 году, в настоящее время стала стандартной. Ее создатели — Йоханнес Браамс (Johannes Braams), Михаэль Гуссенс (Michael Goossens), Алан Джеффри (Alan Jeffrey), Дэвид Карлайл (David Carlisle), Франк Миттельбах (Frank Mittelbach), Крис Роули (Chris Rowley) и Райнер Шёпф (Rainer Schöpf). Иногда встречаются тексты, набранные в более старой версии LaTeX'а (так называемый LaTeX 2.09, описанный в предыдущих изданиях этой книги). Во многих случаях файл, подготовленный в LaTeX'е 2.09, успешно обрабатывается и с помощью LaTeX'а 2_ε, хотя абсолютной совместимости «снизу вверх» все же нет.

1.2. Достоинства и недостатки

Все издательские системы на базе TeX'а обладают достоинствами, заложенными в самом TeX'е. Для новичка их можно описать одной фразой: напечатанный текст выглядит «совсем как в книге». LaTeX как издательская система предоставляет удобные и гибкие средства достичь этого полиграфического качества. В частности, указав с помощью простых средств логическую структуру текста, автор может не вникать в детали оформления, причем эти детали при необходимости нетрудно изменить (чтобы, скажем, сменить шрифт, которым печатаются заголовки, не надо шарить по всему тексту, а достаточно заменить одну строчку в так называемом «стилевом файле»). Такие вещи, как нумерация разделов, ссылки, оглавление и т. п. получаются почти что «сами собой».

Огромным достоинством систем на базе TeX'а является высокое качество и гибкость верстки абзацев и математических формул (в этом отношении TeX до сих пор не превзойден).

Программа TeX (и все издательские системы на ее базе) неприхотлива к технике (в свое время автор этих строк с успехом работал в TeX'е на компьютерах на базе всеми забытого 80286-процессора, и даже на таком электронном тихоходе, как IBM XT, а уж с современными компьютерами проблем заведомо не возникает).

С другой стороны, TeX'овские файлы (особенно английские — с русскими дело обстоит хуже; см. приложение Е) обладают высокой степенью переносимости: вы можете подготовить LaTeX'овский исходный текст на своей IBM PC, переслать его (скажем, по электронной почте) в издательство, и надеяться, что там ваш текст будет правильно обработан и на печати получится в точности то же, что получилось у вас при пробной печати на вашем любимом принтере (с той единственной разницей, что фотонаборный автомат даст отпечаток более высокого качества). Благодаря этому обстоятельству TeX стал очень популярен как язык международного обмена статьями по математике и физике.

Есть у TeX'а и недостатки. Тем, кто привык к редакторам наподобие Word'а, может не понравиться, что TeX не является системой типа WYSIWYG: работа с исходным текстом и просмотр того, как текст будет выглядеть на печати, — разные операции. На взгляд автора, благодаря этой особенности время на подготовку текста типографского качества только сокращается, но представления об удобстве у всех разные.

Далее, хотя параметры оформления менять легко, создать принципиально новое оформление (новый «класс документов») — совсем не простое дело.

Переносимость TeX'овских текстов снижается, если в них предусмотрен импорт графических файлов (эта возможность в TeX'е зависит от его реализации). Впрочем, в последнее время в этом отношении наметился определенный прогресс: стандартом de facto в TeX'овском мире становится представление графики в формате PostScript, причем в LaTeX'е появились удобные средства для включения этой графики в текст.

Наконец, как мы отмечали в предисловии, TeX плохо приспособлен для верстки страниц со сложным взаимодействием текста и графики.

1.3. Литература по TeX'у

Каноническое описание языка TeX и макропакета Plain TeX — трудная книга Дональда Кнута [2]. У рядового пользователя LaTeX'а необходимость читать эту книгу обычно не возникает.

Каноническое описание $\mathcal{AMS}$-TeX'а — книга [5], также написанная самим создателем $\mathcal{AMS}$-TeXа Майклом Спиваком (Michael Spivak).

Наконец, каноническое описание LaTeX'а — книга Лесли Лэмпорта [1]. Настоящее пособие в чем-то у́же, чем книга [1], а в чем-то — шире: мы не упоминаем о некоторых средствах LaTeX'а, которые, на наш взгляд, редко используются на практике, но при этом рассказываем о многих полезных вещах, о которых в [1] не упоминается. Разнообразные модификации стандартного стиля оформления («пакеты») описаны в книге Миттельбаха, Гуссенса и Самарина [6], рассчитанной на читателя, уже умеющего работать в LaTeX'е. Некоторые другие книги по LaTeX'у приведены в списке литературы (с. 447).

Одной из первых книг по LaTeX'у на русском языке была брошюра [4], представляющая собой выполненный А. Шенем перевод краткого руководства [3] с дополнением переводчика, посвященным описанию одной из популярных реализаций TeX'а под DOS — системы emTeX.

1.4. Как проходит работа с системой LaTeX

В дальнейшем мы будем отмечать, какие свойства системы специфичны для LaTeX'а, а какие относятся вообще к TeX'у и ко всем издательским системам на его базе, но при первом чтении вы можете об этих тонкостях не задумываться и воспринимать слова TeX и LaTeX как синонимы.

В частности, все, что сказано в этом разделе, применимо не только к LaTeX'у, но и к любому другому макропакету для TeX'а, хотя мы всегда будем говорить «LaTeX».

Для начала автор должен подготовить с помощью любого текстового редактора файл с текстом, оснащенным командами для LaTeX'а. Такие файлы по традиции имеют расширение `tex` (описанию того, что должно быть в таком файле, и посвящена вся эта книга)[1]. Дальнейшая работа протекает в два этапа. Сначала надо обработать файл с помощью программы-транслятора; в результате получается файл с расширением `dvi` (device independent — не зависящий от устройства).

Теперь полученный файл (его называют еще `dvi`-файлом) можно с помощью программ, называемых `dvi`-драйверами, распечатать на лазерном или струйном[2] принтере, посмотреть на экране (текст будет в таком же виде, как он появится на печати) и т. д. (для разных устройств есть разные драйверы). Неудовлетворенный результатом, автор вносит изменения в исходный файл — и цикл повторяется.

На самом деле повторений цикла будет больше, так как придется еще исправлять синтаксические ошибки в исходном тексте.

Перед тем, как начать работать в системе LaTeX, вам необходимо уяснить для себя три вопроса:

- Что нужно сделать, чтобы оттранслировать исходный текст (т. е. создать из него `dvi`-файл)?
- Что нужно сделать, чтобы просмотреть `dvi`-файл на экране?
- Что нужно сделать, чтобы напечатать `dvi`-файл?

[1] Подчеркнем, что это должен быть именно «чистый» текстовый файл; `doc`-файлы для этих целей не годятся!

[2] Или даже на матричном.

Кроме того, для создания исходного текста нужно, естественно, уметь обращаться с каким-нибудь текстовым редактором.

Ответы на эти вопросы проще всего получить у знакомых, которые уже пользуются TEX'ом; они же помогут его достать (практически все реализации TEX'а являются свободно распространяемыми) и установить. Некоторые сведения о реализациях TEX'а для систем типа Unix (Linux, FreeBSD и др.) и для DOS/Windows приведены в приложении Ж.

2. Основные понятия

2.1. Исходный файл

Исходный файл[3] для системы LATEX представляет собой собственно текст документа вместе со *спецсимволами* и *командами*, с помощью которых системе передаются указания касательно размещения текста. Этот файл можно создать любым текстовым редактором, но при этом необходимо, чтобы в итоге получился так называемый «чистый» текстовый файл. Это означает, что текст не должен содержать шрифтовых выделений, разбивки на страницы и т. п.

Исходный текст документа не должен содержать переносов (TEX сделает их сам). Слова отделяются друг от друга пробелами, при этом TEX не различает, сколько именно пробелов вы оставили между словами (чтобы вручную управлять пробелами, есть специальные команды, о которых пойдет речь позже). Конец строки также воспринимается как пробел. Соседние абзацы должны быть отделены друг от друга пустыми строками (опять-таки все равно, сколько именно пустых строк стоит между абзацами, важно, чтоб была хоть одна).

В правой колонке приведен фрагмент исходного текста, а в левой — то, как он будет выглядеть на печати после обработки системой LATEX.

Слова разделяются пробелами, а абзацы — пустыми строками.

Абзацный отступ в исходном тексте оставлять не надо: он получается автоматически.

```
Слова  разделяются пробелами,
а         абзацы ---
     пустыми строками.

Абзацный отступ в исходном
   тексте   оставлять
         не
      надо: он получается
автоматически.
```

[3] Для нетерпеливых: пример такого файла приведен на с. 21.

Как мог заметить читатель, форматировать исходный текст не обязательно.

Предупреждение: мы используем русские буквы в наших примерах, хотя стандартный LaTeX (без специальной «русификации») их не понимает, и, строго говоря, следовало бы использовать латинские буквы. Подробнее об этом написано в приложении Е.

2.2. Спецсимволы

Большинство символов в исходном тексте прямо обозначает то, что будет напечатано (если в исходном тексте стоит запятая, то и на печати выйдет запятая). Следующие 10 символов:

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

имеют особый статус; если вы употребите их в тексте «просто так», то скорее всего получите сообщение об ошибке (и на печати не увидите того, что хотелось). Печатное изображение знаков, соответствующих первым семи из них, можно получить, если в исходном тексте поставить перед соответствующим символом без пробела знак `\` (по-английски он называется «backslash»):

Курс тугрика повысился на 7%, и теперь за него дают $200.

```
Курс тугрика повысился на
7\%, и теперь за него
дают \$200.
```

Если символ `%` употреблен в тексте не в составе комбинации `\%`, то он является «символом комментария»: все символы, расположенные в строке после него (и сам `%`), TeX игнорирует. С помощью символа `%` в исходный текст можно вносить пометки «для себя»:

Это пример.
Жил-был у бабушки серенький козлик.

```
Это % глупый
% Лучше: поучительный
пример.

Жил-был у бабушки
сере%
  нький козлик.
```

Обратите внимание на предпоследнюю строку: после знака процента игнорируется вся строка, включая ее конец, который в нормальных условиях играет роль пробела; с другой стороны, начальные пробелы в строке игнорируются всегда. Поэтому TeX не видит пробела между кусками слов `сере` и `нький`, и они благополучно складываются в слово «серенький».

Скажем вкратце о смысле остальных спецсимволов. Фигурные скобки ограничивают *группы* в исходном файле (см. с. 22). Знак доллара ограничивает математические формулы (см. разд. 3 и гл. II). При наборе математических же формул используются знаки `_` и `^` («знак подчеркивания» и «крышка»). Знак `~` обозначает «неразрывный пробел» между словами (см. с. 103). Со знака `\` начинаются все TeX'овские команды (см. разд. 2.3). Знаки `#` и `&` используются в более сложных конструкциях TeX'а, о которых сейчас говорить преждевременно.

Отметим еще, что символы `<` `>` `|` в тексте употреблять можно в том смысле, что сообщения об ошибке это не вызовет, но напечатается при этом нечто, совсем на эти символы не похожее. Подлинное место для этих символов, так же как и для символов `=` и `+`, — математические формулы, о которых речь пойдет позже.

2.3. Команды и их задание в тексте

Задание печатного знака процента с помощью последовательности символов `\%` — пример важнейшего понятия TeX'а, называемого *командой*. С точки зрения их записи в исходном тексте, команды делятся на два типа. Первый тип — команды, состоящие из знака `\` и одного символа после него, не являющегося буквой. Именно к этому типу относятся команды `\{`, `\}`,..., `\%`, о которых шла речь на с. 18.

Команды второго типа состоят из `\` и последовательности букв, называемой *именем команды* (имя может состоять и из одной буквы). Например, команды `\TeX`, `\LaTeX` и `\LaTeXe` генерируют эмблемы систем TeX, LaTeX и LaTeX 2_ε. В имени команды, а также между `\` и именем, не должно быть пробелов; имя команды нельзя разрывать при переносе на другую строку.

В именах команд прописные и строчные буквы различаются. Например, `\large`, `\Large` и `\LARGE` — это три разные команды (как вы в дальнейшем узнаете, они задают различные размеры шрифта).

После команды первого типа (из `\` и не-буквы) пробел в исходном тексте ставится или не ставится в зависимости от того, что вы хотите получить на печати:

В чем разница между $1 и $ 1?	`В чем разница между` `\$1 и \$ 1?`

После команды из `\` и букв в исходном тексте *обязательно* должен стоять либо пробел, либо символ, не являющийся буквой (это необходимо, чтобы TeX смог определить, где кончается имя команды и начинается дальнейший текст). Вот примеры с командой `\slshape` (она переключает шрифт на наклонный):

2 turtle doves
and a partridge in a pear tree.

```
\slshape2 turtle doves

\slshape and a partridge
in a pear tree.
```

Если бы мы написали `\slshapeand a partridge...`, то при трансляции TeX зафиксировал бы ошибку (типичную для начинающих) и выдал сообщение о том, что команда `\slshapeand` не определена.

С другой стороны, если после команды из `\` и букв в исходном тексте следуют пробелы, то при трансляции они игнорируются. Если необходимо, чтобы TeX все-таки «увидел» пробел после команды в исходном тексте (например, чтобы сгенерированное с помощью команды слово не сливалось с последующим текстом), надо этот пробел специально организовать. Один из возможных способов — поставить после команды пару из открывающей и закрывающей фигурных скобок `{}` (так что TeX будет знать, что имя команды кончилось), и уже после них сделать пробел, если нужно. Иногда можно также поставить команду `\ ` (backslash с пробелом после него), генерирующую пробел. Вот пример.

Освоить LaTeX проще, чем TeX. Человека, который знает систему TeX и любит ее, можно назвать TeXником.

```
Освоить \LaTeX\ проще,
чем \TeX. Человека,
который знает систему
\TeX{} и любит ее, можно
назвать \TeX ником.
```

В последней строке этого примера мы не создали пробела после команды `\TeX`, чтобы эмблема TeX'а слилась с последующим текстом.

2.4. Структура исходного текста

LaTeX-файл должен начинаться с команды

```
\documentclass
```

задающей стиль оформления документа (в LaTeX'е 2_ε принято говорить «класс документа»).[4]

Пример:

```
\documentclass{book}
```

Слово `book` в фигурных скобках указывает, что документ будет оформлен, как книга: все главы будут начинаться с нечетных страниц, текст будет снабжен колонтитулами некоторого определенного вида и т. п.

[4]В LaTeX 2.09 говорилось `\documentstyle` вместо `\documentclass`.

Кроме класса `book`, в стандартный комплект LaTeX'а входят классы `article` (для оформления статей), `report` (нечто среднее между `article` и `book`), `proc` (для оформления изданий типа «труды конференции») и `letter` (для оформления деловых писем так, как это принято в США). Чтобы задать оформление документа с помощью одного из этих классов, надо в фигурных скобках после команды `\documentclass` указать вместо `book` название требуемого класса. Стандартные классы можно (а иногда и нужно) менять, можно создавать и новые классы, но пока что будем исходить из стандартных классов.

После команды `\documentclass` могут следовать команды, относящиеся ко всему документу и устанавливающие различные параметры оформления текста, например, величину абзацного отступа (вообще-то все эти параметры определяются используемым классом, но может случиться, что вам понадобится сделать в них изменения). Далее должна идти команда

```
\begin{document}
```

Только после этой команды может идти собственно текст. Если вы поместите текст или какую-нибудь команду, генерирующую текст (например, `\LaTeX`) до `\begin{document}`, то LaTeX выдаст сообщение об ошибке. Часть файла, расположенная между командами `\documentclass` и `\begin{document}`, называется преамбулой.

Заканчиваться файл должен командой

```
\end{document}
```

Если после `\end{document}` в файле написано что-то еще, LaTeX это проигнорирует.

Следующий пример показывает LaTeX-файл, составленный по всем правилам. Ничего интересного в результате его обработки не напечатается, но уж зато и сообщений об ошибках вы не получите.

```
\documentclass{article}
\begin{document}
Проба пера.
\end{document}
```

Совет: если вы до сих пор не работали с системой LaTeX на компьютере, сейчас самое время попробовать (только замените русские буквы на латинские, если доступный вам LaTeX не «русифицирован»).

2.5. Группы

Важным понятием TEX'а является понятие *группы*. Чтобы понять, что это такое, рассмотрим пример.

При обработке TEX'ом исходного файла набор текста в каждый момент идет каким-то вполне определенным шрифтом (он называется текущим шрифтом). Изначально текущим шрифтом является «обычный» прямой шрифт (по-английски «roman»). Команда `\slshape`, с которой мы уже столкнулись в разд. 2.3, переключает текущий шрифт на наклонный, а `\upshape` выполняет обратное переключение. Аналогичным образом команды `\bfseries` и `\mdseries` меняют жирность шрифта[5].

Полужирный шрифт начнется с **этого слова. Снова** светлый, теперь *наклонный, до нового переключения; вновь* прямой.

```
Полужирный шрифт начнется
с \bfseries этого слова.
Снова \mdseries светлый,
теперь \slshape наклонный,
до нового переключения;
вновь \upshape прямой.
```

В этом примере можно обойтись и без команд `\mdseries` и `\upshape` (отменяющих действие предыдущих команд). Для этого часть текста, которую вы хотите оформить полужирным или наклонным шрифтом, можно заключить в фигурные скобки, и дать команду `\bfseries` или `\slshape` *внутри* этих скобок! Тогда сразу же после закрывающей фигурной скобки TEX «забудет» про то, что шрифт переключался, и будет продолжать набор тем шрифтом, который был до скобок:

Полужирным шрифтом набрано только **это** слово; после скобок все идет, как прежде.

```
Полужирным шрифтом набрано
только {\bfseries это}
слово; после скобок все
идет, как прежде.
```

Сами по себе фигурные скобки не генерируют никакого текста и не влияют на шрифт; единственное, что они делают — это ограничивают *группу* внутри файла. Как правило, задаваемые командами TEX'а изменения различных параметров (в нашем случае — текущего шрифта) действуют в пределах той группы, внутри которой была дана соответствующая команда; по окончании группы (после закрывающей фигурной скобки, соответствующей той фигурной скобке, что открывала группу) все эти изменения забываются и восстанавливается тот режим, который был до начала группы. Проиллюстрируем все сказанное следующим примером, в котором используется еще команда `\itshape` (она переключает шрифт на курсивный):

[5] Подробнее о переключении шрифтов будет рассказано в разделе III.5.

Сначала переключим шрифт на *курсив; теперь сделаем шрифт еще и* ***полужирным;*** *посмотрите, как восстановится* шрифт после конца группы.

```
Сначала {переключим
шрифт на \itshape курсив;
теперь сделаем шрифт
еще и {\bfseries полужирным;}
посмотрите, как
восстановится} шрифт
после кон{ца  г}руппы.
```

Как видите, группы могут быть вложены друг в дружку. Обратите внимание, что внутри внешней группы курсив начался не с того места, где была открывающая скобка, а только после команды `\itshape` (именно команда, а не скобка, переключает шрифт). Шутки ради мы создали еще одну группу из двух последних буквы слова `конца`, первой буквы слова `группы` и пробела между ними; как и должно быть, на печати это никак не отразилось: ведь внутри скобок мы ничего не делали!

Трюк с постановкой пары скобок `{}` после имени команды, о котором шла речь на с. 20 — тоже пример использования групп. В этом случае скобки ограничивают «пустую» группу; ставятся они в качестве не-букв, ограничивающих имя команды и при этом никак не влияющих на печатный текст.

Фигурные скобки в исходном тексте должны быть сбалансированы[6]: каждой открывающей скобке должна соответствовать закрывающая. Если вы почему-либо не соблюли это условие, при трансляции вы получите сообщение об ошибке.

Некоторые команды, называемые *глобальными,* сохраняют свое действие и за пределами той группы, где они были употреблены. Всякий раз, когда идет речь о глобальной команде, это будет специально оговариваться.

2.6. Команды с аргументами

Команды наподобие `\LaTeX` или, скажем, `\bfseries` действуют «сами по себе»; многим командам, однако, необходимо задать *аргументы.* Первый пример тому дает команда `\documentclass`: слово, указываемое в фигурных скобках, — ее аргумент; если его не указать, то произойдет ошибка. В LaTeX'е аргументы команд бывают обязательные и необязательные. Обязательные аргументы задаются *в фигурных скобках*; если для команды предусмотрено наличие обязательных аргументов, она без них правильно работать не будет. У многих команд предусмотрены также и необязательные аргументы: они влияют на работу команды, коль

[6]Это не относится к скобкам, входящим в состав команд `\{` и `\}`.

скоро они указаны, но их отсутствие не ведет к сообщению об ошибке. Необязательные аргументы задаются *в квадратных скобках*.

В частности, у команды `\documentclass` предусмотрен обязательный аргумент, о котором уже шла речь, и необязательный: в квадратных скобках перед обязательным аргументом можно указать список (через запятую) так называемых классовых опций, т. е. дополнительных особенностей оформления. Например, если мы хотим, чтобы книга набиралась шрифтом кегля 12 вместо кегля 10, принятого по умолчанию[7], и притом в две колонки, мы должны начать файл командой

```
\documentclass[12pt,twocolumn]{book}
```

Есть также классовая опция `11pt`, означающая, что текст будет набираться кеглем 11. Полный список возможных классовых опций приведен в разд. 1 главы IV.

Наряду с классовыми опциями в LaTeX'е используются и так называемые *стилевые пакеты*. После команды `\documentclass`, начинающей файл, может следовать команда `\usepackage`, в аргументе которой стоит (через запятую) список подключаемых этой командой стилевых пакетов. (Можно использовать и несколько команд `\usepackage`.) Например, первые две строки файла могут быть такими:

```
\documentclass[12pt,twocolumn]{book}
\usepackage{amsfonts,longtable}
```

Здесь пакет `amsfonts` подключается, чтобы использовать в математических формулах дополнительные шрифты, позволяющие напечатать что-нибудь вроде $\mathfrak{sl}_2(\mathbb{C})$, а пакет `longtable` нужен, чтобы иметь возможность набирать таблицы, простирающиеся на несколько страниц. Когда мы говорим: «чтобы воспользоваться этой возможностью, необходимо подключить такие-то стилевые пакеты», мы молчаливо предполагаем, что поставка LaTeX'а, которой вы пользуетесь, все эти пакеты содержит. В приложении Ж мы расскажем о том, откуда их взять.

Необязательных аргументов может быть предусмотрено несколько; иногда они должны располагаться до обязательных, иногда после. В любом случае порядок, в котором должны идти аргументы команды, надо строго соблюдать. Между скобками, в которые заключены обязательные аргументы, могут быть пробелы, но не должно быть пустых строк.

2.7. Окружения

Еще одна важная конструкция LaTeX'а — это окружение (environment).

Окружение — это фрагмент файла, начинающийся с текста

[7]Примечание для полиграфистов: TeX'овский кегль 10 примерно соответствует нашему девятому кеглю (см. с. 26).

`\begin{`*имя_окружения*`}`

где *имя_окружения* представляет собой первый обязательный (и, возможно, не единственный) аргумент команды `\begin`. Заканчивается окружение командой

`\end{`*имя_окружения*`}`

(команда `\end` имеет только один аргумент — имя завершаемого ею окружения). Например:

Все строки этого абзаца будут центрированы; переносов не будет, если только какое-то слово, как в дезоксирибону- клеиновой кислоте, не длинней строки.	`\begin{center}` `Все строки этого абзаца будут` `центрированы; переносов не будет,` `если только какое-то слово,` `как в дезоксирибонуклеиновой` `кислоте, не длинней строки.` `\end{center}`

Каждой команде `\begin`, открывающей окружение, должна соответствовать закрывающая его команда `\end` (с тем же именем окружения в качестве аргумента).

Важным свойством окружений является то, что они действуют и как фигурные скобки: *часть файла, находящаяся внутри окружения, образует группу.* Например, внутри окружения `center` в вышеприведенном примере можно было бы сменить шрифт, скажем, командой `\itshape`, и при этом после команды `\end{center}` восстановился бы тот шрифт, что был перед окружением.

2.8. Звездочка после имени команды

В LaTeX'е некоторые команды и окружения имеют варианты, в которых непосредственно после имени команды или окружения ставится звездочка `*`. Например, команда `\section` означает «начать новый раздел документа», а команда `\section*` означает «начать новый раздел документа, не нумеруя его».

2.9. Параметры

Наряду с текущим шрифтом, о котором уже шла речь, TeX в каждый момент обработки исходного текста учитывает значения различных параметров, таких, как величина абзацного отступа, ширина и высота

Таблица I.1. TEX'овские единицы длины

`pt`	пункт ≈ 0.35 миллиметра
`pc`	пика $= 12\,\mathrm{pt}$
`mm`	миллиметр
`cm`	сантиметр $= 10\,\mathrm{mm}$
`in`	дюйм $= 25{,}4\,\mathrm{mm}$
`dd`	пункт Дидо $\approx 1{,}07\,\mathrm{pt}$
`cc`	цицеро $= 12\,\mathrm{dd}$

страницы, расстояние по вертикали между соседними абзацами, а также великое множество других важных вещей. Расскажем, как можно менять эти параметры, если это понадобится.

Параметры TEX'а обозначаются аналогично командам: с помощью символа `\` («backslash»), за которым следует либо последовательность букв, либо одна не-буква. Например, `\parindent` обозначает в TEX'е величину абзацного отступа; если нам понадобилось, чтобы абзацный отступ равнялся двум сантиметрам, можно написать так:

```
\parindent=2cm
```

(Это изменение распространяется лишь на текущую группу: после конца группы восстановится старое значение отступа.)

Аналогично поступают и в других случаях: чтобы изменить параметр, надо написать его обозначение, а затем, после знака равенства, значение, которое мы «присваиваем» этому параметру; в зависимости от того, что это за параметр, это может быть просто целое число, или длина (как в нашем примере), или еще что-нибудь.

2.10. Единицы длины

Многие параметры, используемые TEX'ом, являются размерами (пример тому мы видели в разд. 2.9); в табл. I.1 собраны единицы длины (кроме нескольких экзотических), которые можно использовать в TEX'е при задании размеров.

Замечание для полиграфистов: TEX'овский пункт является единицей измерения, принятой в англо-американской типометрии; он отличается от пункта, принятого в континентальной Европе (в том числе и в России). Единица измерения, называемая в TEX'е пунктом Дидо, соответствует пункту, к которому привыкли отечественные полиграфисты.

Можно задавать размеры с помощью любой из этих единиц; при записи дробного числа можно использовать как десятичную запятую, так и десятичную точку (в таблице мы использовали оба способа); прописные и строчные буквы в обозначениях единиц длины не различаются.

Даже если длина, которую вы указываете TeX'у, равна нулю, все равно необходимо указать при этом нуле какую-нибудь из используемых TeX'ом единиц длины. Например, если написать

```
\parindent=0
```

то вы получите сообщение об ошибке; вместо `0` надо было бы писать, например, `0pt` или `0in`.

Кроме перечисленных, в TeX'е используются еще две «относительные» единицы длины, размер которых зависит от текущего шрифта. Это `em`, приблизительно равная ширине буквы «M» текущего шрифта, и `ex`, приблизительно равная высоте буквы «x» текущего шрифта. Эти единицы удобно использовать в командах, которые должны работать единообразно для шрифтов разных размеров.

2.11. Автоматическая генерация ссылок

Система LaTeX предоставляет возможность организовать ссылки на отдельные страницы или разделы документа таким образом, чтобы программа сама определяла номера страниц или разделов в этих ссылках. Объясним это на примере.

Представим себе, что вам нужно сослаться на какое-то место в вашем тексте. Проще всего было бы указать страницу, на которой это место находится, написав «...как мы уже отмечали на с. 99» или что-то в этом роде, но как угадать, на какую страницу печатного текста попадет это место? Вместо того, чтобы гадать, можно сделать следующее:

- пометить то место, на которое вы хотите сослаться в дальнейшем (или предшествующем) тексте;
- в том месте текста, где вы хотите поместить ссылку, поставить команду-ссылку на вашу метку.

Конкретно это реализуется так. Помечается любое место текста с помощью команды `\label`. Эта команда имеет один обязательный аргумент (помещаемый, стало быть, в фигурных скобках) — «метку». В качестве метки можно использовать любую последовательность букв и цифр (не содержащую пробелов). Например, эта команда может иметь вид

```
\label{wash}
```

Ссылка на страницу, на которой расположена метка, производится с помощью команды `\pageref`. У нее также один обязательный аргумент — та самая метка, на которую вы хотите сослаться. Пример:

Обязательно мойте руки перед едой, чтобы не заболеть.

Как известно (см. с. 99), руки надо мыть.

```
Обязательно мойте
руки\label{wash} перед
едой, чтобы не заболеть.

Как известно (см.
с.~\pageref{wash}),
руки надо мыть.
```

Обратите внимание, что мы поставили команду `\label` рядом с ключевым словом «руки» без пробела, чтобы гарантировать, что будет помечена именно та страница, на которую попало это слово.

В этом примере мы использовали еще значок `~` («неразрывный пробел»), чтобы при печати сокращение «с.» попало на ту же строку, что и номер страницы. Подробности см. в разд. III.3.1.

После того как вы впервые вставите в свой файл команды `\label` и `\pageref`, при трансляции вы получите сообщение о том, что ваша ссылка не определена, а на печати или при просмотре увидите на месте своих ссылок вопросительные знаки. Дело в том, что в этот момент LaTeX еще не знает значения ваших меток: он только записывает информацию о них в специальный файл (с тем же именем, что у обрабатываемого файла, и расширением `aux`); при следующем запуске он прочтет эту информацию и подставит ссылки. Если в промежутке между двумя запусками в файл были внесены изменения, это может привести к сдвигу нумерации страниц. Если такие изменения действительно произошли, LaTeX сообщит вам об этом и попросит запустить программу еще раз, чтобы получить корректные ссылки. (На практике иногда нужно и больше запусков программы LaTeX — если в книге есть оглавление, предметный указатель и др.)

Если вы после двух запусков подряд получите сообщение о неопределенной ссылке, значит, в исходном тексте присутствует ошибка (вероятнее всего, опечатка в аргументе команды `\pageref`; возможно, вы забыли включить в текст команду `\label`).

На место, помеченное с помощью команды `\label`, можно сослаться с помощью команды `\ref`, а не `\pageref` — тогда на печати получится не номер страницы, а номер раздела документа, в котором находится метка, или номер рисунка, или номер элемента в «нумерованном перечне и т. п. — пометить с возможностью ссылки можно почти любой элемент документа. Об этом мы будем подробно говорить в разд. IV.9.

3. Набор формул в простейших случаях

3.1. Основные принципы

В документе, подготовленном с помощью TeX'а, различают математические формулы внутри текста и «выключные» (выделенные в отдельную строку). Формулы внутри текста окружаются знаками `$` (с обеих сторон). Выключные формулы окружаются парами знаков доллара `$$` с обеих сторон. Формулами считаются как целые формулы, так и отдельные цифры или буквы, в том числе греческие, а также верхние и нижние индексы и спецзнаки. Пробелы внутри исходного текста, задающего формулу, игнорируются (но по-прежнему надо ставить пробелы, обозначающие конец команды): TeX расставляет пробелы в математических формулах автоматически (например, знак равенства окружается небольшими пробелами). Пустые строки внутри текста, задающего формулу, не разрешаются. Если нужен пробел до или после внутритекстовой формулы, надо оставить его вне долларов. То же самое относится и к знакам препинания, следующим за внутритекстовой формулой: их также надо ставить после закрывающего формулу знака доллара. (В выключных формулах приходится указывать знаки препинания внутри долларов, иначе эти знаки попадут на следующую строку.) Каждая буква в формуле рассматривается как имя переменной и набирается шрифтом «математический курсив» (в отличие от обычного курсива, в нем увеличены расстояния между соседними буквами).

Часть файла, составляющая математическую формулу, образует группу (см. с. 22): изменения параметров, произведенные внутри формулы, забываются по ее окончании.

3.2. Степени и индексы

Степени и индексы набираются с помощью знаков `^` и `_` соответственно.

Катеты a, b треугольника связаны с его гипотенузой c формулой $c^2 = a^2 + b^2$ (теорема Пифагора).	`Катеты $a$, $b$ треугольника связаны с его гипотенузой $c$ формулой $c^2=a^2+b^2$ (теорема Пифагора).`

Если индекс или показатель степени — выражение, состоящее более чем из одного символа, то его надо взять в фигурные скобки:

Из теоремы Ферма следует, что уравнение

$$x^{4357} + y^{4357} = z^{4357}$$

не имеет решений в натуральных числах.

```
Из теоремы Ферма следует,
что уравнение
$$
  x^{4357}+y^{4357}=z^{4357}
$$
не имеет решений в
натуральных числах.
```

Если у одной буквы есть как верхние, так и нижние индексы, то можно указать их в произвольном порядке:

Обозначение R^i_{jkl} для тензора кривизны было введено еще Эйнштейном.

```
Обозначение $R^i_{jkl}$ для
тензора кривизны было
введено еще Эйнштейном.
```

Если же требуется, чтобы верхние и нижние индексы располагались не друг под другом, а на разных расстояниях от выражения, к которому они относятся, то нужно немного обмануть TEX, оформив часть индексов как индексы к «пустой формуле» (паре из открывающей и закрывающей скобок):

Можно также написать $R_j{}^i{}_{kl}$, хотя не всем это нравится.

```
Можно также написать
$R_j{}^i{}_{kl}$, хотя
не всем это нравится.
```

Если вы хотите написать формулу, читающуюся как «два в степени, равной икс в кубе», то запись `2^x^3` вызовет сообщение об ошибке; правильно будет `2^{x^3}` (на печати это будет выглядеть как 2^{x^3}).

3.3. Дроби

Дроби, обозначаемые косой чертой (так рекомендуется обозначать дроби во внутритекстовых формулах), набираются непосредственно:

Неравенство $x+1/x \ge 2$ выполнено для всех $x > 0$.

```
Неравенство $x+1/x\ge 2$
выполнено для всех $x>0$.
```

В этом примере мы еще использовали знаки «строгих» неравенств (в TEX'овских формулах они набираются непосредственно, как знаки `>` и `<`) и нестрогих неравенств (знак «больше или равно» генерируется командой `\ge`, «меньше или равно» — командой `\le`). Если вы употребите символы `<` и `>` в обычном тексте, вне формул, то вместо знаков «меньше» и «больше» увидите небольшой сюрприз.

Наряду со знаками для нестрогих неравенств, TEX предоставляет большое количество специальных символов для математических формул (греческие буквы также рассматриваются как специальные символы). Все эти символы набираются с помощью специальных команд (не требующих параметров). Списки этих команд вы найдете в таблицах в начале следующей главы.

Если вы используете в формуле десятичные дроби, в которых дробная часть отделена от целой с помощью запятой, то эту запятую следует взять в фигурные скобки (в противном случае после нее будет оставлен небольшой дополнительный пробел, что нежелательно):

$\pi \approx 3{,}14$

```
$\pi\approx 3{,}14$
```

Здесь команда `\pi` порождает греческую букву π, а команда `\approx` дает знак $\approx$ («приближенно равно»). Десятичную точку в дробях заключать в скобки не нужно.

Дроби, в которых числитель расположен над знаменателем, набираются с помощью команды `\frac`. Эта команда имеет два обязательных аргумента: первый — числитель, второй — знаменатель. Пример:

$$\frac{(a+b)^2}{4}-\frac{(a-b)^2}{4}=ab$$

```
$$
  \frac{(a+b)^2}{4}-
  \frac{(a-b)^2}{4}=
  ab
$$
```

Если числитель и/или знаменатель дроби записывается одной буквой (в том числе греческой) или цифрой, то можно их и не брать в фигурные скобки:

$$\frac12+\frac x 2=\frac{1+x}2$$

```
$$\frac12+\frac x 2=
\frac{1+x}2$$
```

3.4. Скобки

Круглые и квадратные скобки набираются как обычно, для фигурных скобок используются команды `\{` и `\}`, для других также есть специальные команды (см. следующую главу).

Команда `\left` перед открывающей скобкой в совокупности с командой `\right` перед соответствующей ей закрывающей скобкой позволяет автоматически выбрать нужный размер скобки:

$$1+\left(\frac{1}{1-x^{2}}\right)^3$$

```
$$
  1+\left(\frac{1}{1-x^{2}}
    \right)^3
$$
```

Подробнее о скобках, размер которых выбирается автоматически, рассказано в следующей главе (разд. II.2.5).

3.5. Корни

Квадратный корень набирается с помощью команды `\sqrt`, обязательным аргументом которой является подкоренное выражение; корень произвольной степени набирается с помощью той же команды `\sqrt` с необязательным аргументом — показателем корня (необязательный аргумент у этой команды ставится перед обязательным). Пример:

По общепринятому соглашению, $\sqrt[3]{x^3}=x$, но $\sqrt{x^2}=|x|$.

```
По общепринятому соглашению,
$\sqrt[3]{x^3}=x$, но
$\sqrt{x^2}=|x|$.
```

Обратите внимание, что вертикальные черточки, обозначающие знак модуля, набираются непосредственно.

3.6. Штрихи и многоточия

Штрихи в математических формулах обозначаются знаком `'` (и *не* оформляются как верхние индексы):

Согласно формуле Лейбница,

$$(fg)''=f''g+2f'g'+fg''.$$

Это похоже на формулу квадрата суммы.

```
Согласно формуле Лейбница,
$$
  (fg)''=f''g+2f'g'+fg''.
$$
Это похоже на формулу
квадрата суммы.
```

Если надо записать формулу, читающуюся как «x штрих в квадрате», возьмите `x'` в фигурные скобки, вот так: `${x'}^2$`. Обратите еще внимание на то, что точку мы поставили в конце выключной формулы (если бы мы поставили ее после знаков `$$`, то с нее начался бы абзац, следующий после формулы).

В математических формулах встречаются многоточия; TeX различает многоточие, расположенное внизу строки (обозначается `\ldots`), и расположенное по центру строки (оно обозначается `\cdots`). Первое из них используется при перечислениях, второе — когда нужно заменить пропущенные слагаемые или сомножители (такова американская традиция; в России обычно многоточие ставят внизу строки и в этом случае):

В детстве К.-Ф. Гаусс придумал, как быстро найти сумму

$$1+2+\cdots+100=5050;$$

это случилось, когда школьный учитель задал классу найти сумму чисел $1,2,\ldots,100$.

```
В~детстве К.-Ф.~Гаусс придумал,
как быстро найти сумму
$$
  1+2+\cdots+100=5050;
$$
это случилось, когда школьный
учитель задал классу найти
сумму чисел $1,2,\ldots,100$.
```

Знак ~ после инициалов великого Гаусса мы поставили, чтобы фамилия не могла перенестись на другую строку отдельно от инициалов (см. с. 103). В LaTeX'е можно использовать команду `\ldots` и в обычном тексте, вне математических формул, для знака многоточия (см. с. 100).

3.7. Имена функций

Функции наподобие sin, log и т. п., имена которых принято печатать прямым шрифтом, набираются с помощью специальных команд (обычно одноименных с обозначениями соответствующих функций).

Как знают некоторые школьники, $\log_{1/16} 2=-1/4$, а $\sin(\pi/6)=1/2$.

```
Как знают некоторые школьники,
$\log_{1/16}2=-1/4$, а
$\sin(\pi/6)=1/2$.
```

Заметьте, что основание логарифма задается как нижний индекс.

Полный список команд, аналогичных `\log` или `\sin`, приведен в разд. II.1.2.

В стандартный набор команд TeX'а не входят команды для функций tg и ctg (в англоязычных странах эти функции принято обозначать tan и cot соответственно). Большой беды тут нет, поскольку эти недостающие команды легко определить самому (см. гл. II).

4. Разбиение исходного файла на части

Команды, рассматриваемые в этом разделе, помогают разумно организовать исходный текст.

Часто бывает удобно разбить большой текст на несколько частей, хранящихся в разных файлах. Чтобы можно было объединить их в одно целое, в LaTeX'е предусмотрена команда `\input`. Если в тексте написать

`\input{`*имя_файла*`}`

то TeX будет работать так, как если бы вместо строки с командой `\input` стоял текст файла, имя которого вы указали.

Обычно, готовя текст большого объема, создают небольшой «головной файл», в котором между `\begin{document}` и `\end{document}` размещены строки с командами `\input`, задающими включение файлов, в которых и записана основная часть текста. Например, книгу из четырех глав, записанных в файлах `ch1.tex`,…, `ch4.tex`, можно организовать в виде файла из девяти строчек (именно его, а не файлы с отдельными главами, надо будет передать для обработки LaTeX'у):

```
\documentclass[11pt]{report}
\frenchspacing
\pagestyle{plain}
\begin{document}
\input{ch1}
\input{ch2}
\input{ch3}
\input{ch4}
\end{document}
```

(Если расширение файла, являющегося аргументом команды `\input`, не указано, то LaTeX по умолчанию добавляет расширение `.tex`.)

Ради реализма мы в этом примере включили в преамбулу парочку команд, которые могли бы там появиться и на практике. Первая из них означает, что после знаков препинания пробелы не увеличиваются, как это и принято в России и континентальной Европе (по умолчанию эти пробелы увеличиваются в соответствии с англо-саксонской традицией), а вторая — что номера страниц будут печататься снизу и при этом колонтитулов не будет. Позже мы рассмотрим эти вещи подробнее.

Если в вашем тексте присутствуют команды `\input`, то в процессе трансляции при начале чтения соответствующего файла на экран выдается его имя, чтобы вы понимали, к какому из ваших файлов будут относиться дальнейшие сообщения TeX'а (если таковые будут).

Если вы хотите, чтобы TeX прочитал только часть вашего файла, можно воспользоваться командой без параметров `\endinput`. Если она присутствует в файле, то файл будет прочитан только до строчки, в которой написано `\endinput`, после чего его чтение прекратится.

Если фрагменты текста, включаемые с помощью команд `\input`, должны на печати начинаться с отдельной страницы (например, если это главы книги, как в приведенном примере), то удобно вместо `\input` воспользоваться командой `\include` (ее единственный обязательный аргумент — имя включаемого файла). Выгода здесь в том, что при пользовании командой `\include` можно в процессе работы над текстом попросить LaTeX обрабатывать только часть включаемых файлов.

Для этого надо добавить в преамбулу команду `\includeonly`, в аргументе которой приведен (через запятые) список обрабатываемых файлов. Пусть, скажем, в вышеприведенном примере работа над первой главой уже завершена, за четвертую главу вы еще не принялись, а вторую и третью уже вовсю редактируете. Тогда головной файл можно организовать так:

```
\documentclass[11pt]{report}
\frenchspacing
\pagestyle{plain}
\includeonly{ch2,ch3}
\begin{document}
\include{ch1}
\include{ch2}
\include{ch3}
\include{ch4}
\end{document}
```

(Внимание! В аргументе команды `\include` расширение `.tex` опускать необходимо! Файлы с расширениями, отличными от `.tex`, с помощью этой команды подключать нельзя.) Когда вы перейдете к работе над другими главами, аргумент команды `\includeonly` надо будет соответствующим образом изменить, а когда весь текст будет готов, можно вообще удалить `\includeonly` из файла. При этом количество запусков LaTeX'а, необходимых для получения правильных ссылок, задаваемых командами `\ref` и `\pageref`, будет меньше, чем при проведении аналогичных манипуляций с `\input`.

Комаду `\include` нельзя употреблять в файле, который сам включается в текст с помощью `\include` (для `\input` такого запрета нет).

5. Обработка ошибок

В исходных текстах для TeX'а, которые вы будете готовить, неизбежно будут присутствовать ошибки. В настоящем разделе мы обсудим, как TeX на них реагирует и как вам, в свою очередь, следует реагировать на эти реакции TeX'а.

Все сообщения, которые TeX выдает на экран в процессе трансляции исходного текста, все ваши ответы на эти сообщения, вообще все, что в процессе трансляции появляется на экране, записывается в специальный файл — протокол трансляции. Обычно файл-протокол имеет то же имя, что обрабатываемый TeX'ом файл, и расширение `log`, поэтому он называется `log`-файлом. Когда трансляция будет завершена, вы можете в спокойной обстановке просмотреть log-файл и проанализировать, что произошло.

Часть информации, выдаваемой при трансляции на экран и в log-файл, представляет собой предупреждения (например, о нештатных ситуациях при верстке абзаца), при выдаче которых трансляция не прерывается (в разд. III.6 будет подробно рассказано, что означают такие предупреждения). Если, однако, TEX натыкается на синтаксическую ошибку в исходном тексте, трансляция приостанавливается, а на экран выдается сообщение об ошибке.

Чтобы понять, что делать с этими сообщениями, проведем эксперимент. Наберите следующий файл test.tex из 14 строк, в котором умышленно допущено несколько ошибок (только не сделайте лишних ошибок при наборе):

```
\documentclass{article}
\begin{document}
По-английски специалист по \TeX'у называется \TeXpert.
Следующая строка будет центрирована:
\begin{center}
Строка в центре.
\end{centrr}
А теперь попробуем формулы, например, такие,
как (2x+1)^3=5x$. И еще выключную формулу:
$$\frac{25}{36}=\lrft(\frac{1}
   {1+\frac{1}{5}}\right)^2.
$$
И последняя формула: $\sqrt{4 = 2$.
\end{document}
```

Теперь обработайте наш файл test.tex с помощью LATEX'а. Вскоре вы увидите на экране вот что:

```
! Undefined control sequence.
l.3 ...алист по \TeX'у называется \TeXpert

                                           .
?
```

Первая строка TEX'овского сообщения об ошибке всегда начинается с восклицательного знака, после которого идет краткое указание на характер ошибки (в нашем случае речь идет о том, что обнаружена несуществующая команда). Второй обязательный элемент сообщения об ошибке — строка, начинающаяся с l., после которой идет номер строки исходного текста с ошибкой (в нашем случае 3). После номера на экран выдается сама эта строка или та ее часть, которую TEX успел прочесть к моменту обнаружения ошибки. В нашем случае текст был прочитан до несуществующей команды `\TeXpert` включительно (эта «команда» получилась потому, что мы забыли оставить пробел, ограничивающий имя команды `\TeX`, — см. с. 19), на которой TeX и прервал чтение файла. Наконец, третий основной элемент сообщения об ошибке — строка, состоящая из одного вопросительного знака. Этот вопросительный знак представляет собой «приглашение» пользователю: вам теперь предстоит на сообщение об ошибке отреагировать. Рассмотрим возможные реакции.

Во-первых, всегда можно нажать клавишу `x` или `X` (латинскую) и после этого «ввод» (“Enter”): тогда трансляция немедленно завершится. Может быть, именно так стоит поступать начинающему TeXнику, чтобы разбираться с ошибками по очереди. Но можно и просто нажать клавишу «ввод»: при этом TeX исправит обнаруженную ошибку «по своему разумению» и продолжит трансляцию. Догадаться о том, что ошибка произошла именно из-за забытого пробела, программа, естественно, не может: исправление будет заключаться попросту в том, что будет проигнорирована несуществующая команда `\TeXpert` (так что из печатного текста будет неясно, как по-английски называют специалиста по TeX'у). Нажимать «ввод» в ответ на сообщения об ошибках — довольно распространенная на практике реакция. Если вы твердо намерены нажимать на «ввод» в ответ на все сообщения об ошибках, то можно в ответ на первое же из этих сообщений нажать на `S` или `s`, а затем на «ввод»; при обнаружении дальнейших ошибок трансляция прерываться не будет (TeX будет обрабатывать ошибки так, как если бы вы все время нажимали на «ввод»), по экрану пронесутся сообщения об ошибках, а затем вы сможете их изучить, просмотрев `log`-файл.

Итак, трансляция продолжается. Следующая остановка будет с таким сообщением:

```
! LaTeX Error: \begin{center} on input line 5
                                        ended by \end{centrr}.
See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...
l.7 \end{centrr}

?
```

Это сообщение об ошибке начинается со слов `LaTeX Error`. Такого рода сообщения не встроены в TeX, а создаются LaTeX'ом. В них также присутствуют строка, начинающаяся с `!`, строка, начинающаяся с `l.`, и приглашение — вопросительный знак. Есть на экране и объяснение ошибки: из-за опечатки (`centrr` вместо `center`) получилось, что команда `\begin`, открывающая окружение, не соответствует команде `\end`, закрывающей его (см. разд. 2.7: имена окружений при открывающем `\begin` и закрывающем `\end` должны совпадать). Так или иначе, давайте снова нажмем на «ввод»; тут же мы увидим вот что:

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   ^
l.9 как (2x+1)^
               3=5x$. И~еще выключную формулу:
```

На сей раз мы забыли знак доллара, открывающий формулу; TEX, однако, понял это не сразу, а лишь наткнувшись на символ `^`, который вне формул таким образом использовать нельзя. Нажмем «ввод»: TEX исправит положение, вставив знак доллара непосредственно перед знаком `^`, и пойдет дальше (все такие исправления не вносятся в ваш файл, а происходят только в оперативной памяти компьютера). На печати формула будет иметь странный вид, поскольку (2x+1) будет набрано прямым шрифтом, а 5x — курсивным, но TEX сможет продолжить трансляцию (и искать дальнейшие ошибки).

Следующая ошибка будет уже знакомого нам типа, только на сей раз несуществующая команда получается не из-за забытого пробела, а из-за опечатки (`\lrft` вместо `\left`):

```
! Undefined control sequence.
l.10 $$\frac{25}{36}=\lrft
                          (\frac{1}
?
```

Нажав очередной раз на «ввод», мы немедленно увидим сообщение еще об одной ошибке:

```
! Extra \right.
l.11    {1+\frac{1}{5}}\right)
                              ^2
?
```

Откуда это, ведь в строке 11 у нас все правильно?! Оказывается, эта ошибка была наведена предыдущей. В самом деле, перед этим TEX проигнорировал «команду» `\lrft`, набранную вместо `\left` (именно так

TeX и делает, если в ответ на ошибку «несуществующая команда» нажать на клавишу «ввод»), так что команду `\left` TeX вообще не видел; теперь выходит так, что в тексте, который видит TeX, присутствует `\right` без `\left`, что запрещено (см. разд. II.2.5). Ввиду возможности появления таких «наведенных» ошибок, исправлять ошибки надо, начиная с самой первой; не исключено, что при ее исправлении часть последующих пропадет сама собой.

Нажмем на «ввод» и на этот раз; TeX опять по-свойски исправит ошибку, и вскоре вы увидите такое сообщение:

```
! Missing } inserted.
<inserted text>
                }
<to be read again>
                   $
l.13 И~последняя формула: $\sqrt{4 = 2$
                                        .
?
```

На сей раз ошибка в том, что мы забыли закрывающую фигурную скобку. Нажмем на «ввод»; TeX вставит недостающую скобку (в результате чего на печати получится забавная формула $\sqrt{4=2}$, соответствующая тексту `$\sqrt{4=2}$`: пропажа закрывающей скобки обнаружилась не там, где мы ее забыли, а там, где ее отсутствие вошло в противоречие с синтаксическими правилами TeX'а), после чего трансляция наконец завершится. Кстати, цифра 1 в квадратных скобках, появляющаяся при этом на экране, означает, что TeX сверстал страницу номер 1 и записал ее содержимое в `dvi`-файл. Теперь можно и просмотреть, как будет выглядеть наш текст на печати.

Количество различных сообщений об ошибках, которые может выдавать TeX, составляет несколько сотен, и нормальная реакция на них обычно такая же, как в нашем эксперименте; сейчас мы рассмотрим еще две типичные ошибки, реакция на которые должна быть иной.

Во-первых, может случиться, что в качестве аргумента команды `\input` задано имя несуществующего файла. В этом случае вы получите сообщение наподобие следующего:

```
! LaTeX Error: File 'ttst.tex' not found.

Type X to quit or <RETURN> to proceed,
or enter new name. (Default extension: tex)

Enter file name:
```

В ответ на это следует набрать правильное имя файла и нажать на «ввод», и трансляция благополучно продолжится. Если вообще никакого файла нет (например, TEX запущен по ошибке), наберите `null` — это всегда существующий пустой файл. (В некоторых версиях — `nul` с одним `l`.)

Можно отреагировать на эту ошибку и так же, как на любую другую: нажать `x` и «ввод» (трансляция прервется) или `s` и «ввод» (неправильная команда `\input` будет проигнорирована, на дальнейшие ошибки TEX будет реагировать так, как если бы вы все время нажимали «ввод»). А если вы работаете с LATEX'ом под операционной системой UNIX или каким-то ее аналогом (Linux, например), то в ответ на сообщение о такой ошибке можно и попросту нажать «ввод», и она будет проигнорирована.

Если команда `\input` с именем несуществующего файла попадется TEX'у после того, как вы в ответ на какую-то из прежних ошибок сказали `s`, то трансляция на этом месте тем не менее остановится и TEX поинтересуется верным именем файла.

Вторая ошибка, о которой мы хотели сказать, строго говоря, ошибкой не является; скорее, это нештатная ситуация. Чтобы смоделировать ее, проведем такой эксперимент: удалим из нашего файла `test.tex` последнюю строчку, гласящую `\end{document}`, и снова запустим LATEX для обработки этого файла. Нажав сколько-то раз «ввод», мы обнаружим, что работа TEX'а не закончилась, а на экран выдана звездочка: `*`. Эта звездочка — приглашение TEX'а ввести еще текст или команды; она появляется, когда в исходном тексте отсутствует команда для TEX'а «завершить работу» (в LATEX'е эта команда входит в качестве составной части в комплекс действий, выполняемых командой `\end{document}`). Теперь можно вводить с клавиатуры любой текст и команды — TEX отреагирует на них так же, как если бы этот текст и команды присутствовали в вашем файле. Не будем баловаться, а просто наберем `\end{document}` и нажмем на «ввод», после чего трансляция благополучно завершится. Вряд ли вы будете очень часто забывать последнюю строчку в исходном тексте, но иногда, в результате какой-либо сложной ошибки, может случиться так, что TEX «не заметит» строки `\end{document}`, и вот тогда-то вы и окажетесь лицом к лицу с TEX'овским приглашением-звездочкой.

Бывают и такие хитрые ошибки, что `\end{document}` в ответ на приглашение-звездочку LATEX не удовлетворяет и на экране снова появляется звездочка. На этот случай в LATEX'е предусмотрено последнее средство: команда `\stop`. Если вы введете ее в ответ на TEX'овское приглашение, то, скорее всего, трансляция все-таки прервется. Если и `\stop` не помогает, остается только перезагрузить компьютер или «убить процесс» более цивилизованным способом.

Скажем еще об одном нередко встречающемся TeX'овском сообщении. Если вы открыли группу с помощью фигурной скобки, но забыли ее закрыть, то, даже если трансляция не будет прерываться, в конце вы заведомо получите такое TeX'овское предупреждение:

```
(\end occurred inside a group at level 1)
```

(вместо `1` может стоять и другая цифра, в зависимости от того, сколько вложенных групп вы забыли закрыть). В частности, такое сообщение будет, если вы забыли закрыть или неправильно закрыли какое-то LaTeX'овское окружение (но в этом случае LaTeX выдаст вам и свое сообщение об ошибке, как в нашем примере с `\begin{center}` и `\end{centrr}`).

Наряду с пассивной реакцией на ошибки — все время нажимать на «ввод» или сказать `s` — есть и другая возможность: прямо с клавиатуры вносить исправления в тот текст, который «видит» TeX. На содержимое файла это не повлияет, но изменения в файл можно будет внести и позднее, руководствуясь тем, что записано в `log`-файле. При этом может сэкономиться время за счет того, что будет меньше «наведенных» ошибок и, как следствие, потребуется меньше прогонов TeX'а для отладки.

Чтобы внести исправления с клавиатуры, надо нажать `i` или `I` и затем «ввод». На экране появится такое приглашение:

```
insert>
```

В ответ на это приглашение следует ввести тот текст и/или команды, которые вы хотите вставить в текст, читаемый TeX'ом. Чтобы продемонстрировать это на практике, давайте приведем файл `test.tex` в исходное состояние, вернув в него строку `\end{document}`, и еще раз запустим LaTeX для его обработки. В ответ на первое же сообщение (по поводу несуществующей команды `\TeXpert`) нажмем `i` , а затем, в ответ на приглашение `insert>`, наберем правильный текст

```
\TeX pert
```

и нажмем на «ввод». В ответ на вторую ошибку (когда мы в команде `\end` допустили опечатку в имени окружения `center`) скажем сначала `i`, а затем (в ответ на приглашение) `\end{center}` (кстати, можно делать такие вещи и в один шаг: сразу набрать `i\end{center}` и нажать «ввод»). В ответ на следующую ошибку ничего не остается, как по-прежнему нажать на «ввод»: те символы в исходном тексте, между которыми должен был стоять пропущенный знак доллара, уже поглощены TeX'ом, и вставить его куда надо в данный момент невозможно; зато в ответ на следующую ошибку (`\lrft` вместо `\left`) наберем `i\left` и нажмем на «ввод». Следующей («наведенной») ошибки вообще не будет (ведь на сей раз в тексте, который видит TEX, команда `\left` присутствует, а поэтому и на команду `\right` он отреагирует правильно); наконец, в ответ на последнюю ошибку опять ничего не остается, кроме как нажать на «ввод»: вставить закрывающую фигурную скобку между `4` и знаком равенства прямо с клавиатуры невозможно. Теперь можно просмотреть, как на сей раз будет выглядеть на печати наш текст; некоторые несуразности наподобие $\sqrt{4=2}$ в нем останутся, но их будет меньше, чем если бы мы нажимали на «ввод»: не будет потеряно слово «TEXpert», центрированная строка будет действительно центрирована, формула

$$\frac{25}{36}=\left(\frac{1}{1+\frac{1}{5}}\right)^2$$

будет выглядеть так, как надо. Теперь остается внести исправления в исходный файл (справляясь с тем, что записано в `log`-файле) и запустить LATEX вторично, чтобы получить безошибочный текст.

Как мы уже отмечали, в ответ на сообщение об ошибке всегда можно прервать трансляцию, нажав `X` или `x` и «ввод»; кроме того, бывают случаи, когда TEX прерывает трансляцию «по своей инициативе». На практике важны два случая:

- TEX обнаружил 100 ошибок в пределах одного абзаца — тогда выдается сообщение

  ```
  (That makes 100 errors; please try again.)
  ```

- TEX'у не хватило памяти — тогда выдается сообщение типа

  ```
  ! TeX capacity exceeded, sorry [main memory size=263001].
  ```

Нехватка памяти может возникнуть в результате таких ошибок, из-за которых TEX «зацикливается»; тогда достаточно исправить ошибку. Иногда памяти может действительно не хватить. Так бывает, если в тексте встречаются чудовищно длинные абзацы[8] или сверхсложные таблицы с очень большим количеством строк и столбцов (см. гл. VI по поводу таблиц). Если вы встретились с такой проблемой, то можно проконсультироваться со специалистом (или самому изучить по книге [2]),

[8]На моем домашнем компьютере TEX'овская память иссякла, когда длина абзаца превысила 34 страницы.

как использовать TEX более эффективно (в частности, TEX можно научить переваривать сколь угодно длинные абзацы). Можно также попробовать найти транслятор TEX'а, дающий возможность работать с увеличенным объемом памяти.

Скажем пару слов про более редкие способы реакции на ошибки. Во-первых, в ответ на приглашение `?` можно набрать `h` или `H` и нажать «ввод». В этом случае TEX выдаст на экран дополнительную информацию по поводу вашей ошибки (вряд ли вы много из нее почерпнете, если вы не TEXник), а затем еще раз приглашение `?`. Во-вторых, можно набрать `r` или `R` (и «ввод», естественно); результат будет такой же, как если бы вы сказали `s`, с той разницей, что в случае, когда аргументом команды `\input` служит несуществующий файл, никаких вопросов задаваться не будет, а трансляция просто прервется. Наконец, можно набрать `Q` или `q` (и «ввод»): результат будет такой же, как от `R`, с той разницей, что на экран не будет выдаваться вообще ничего (в `log`-файл все будет записано).

Наконец, режимы реакции на ошибки, задаваемые с клавиатуры с помощью клавиш `s`, `r` или `q`, можно задать прямо в файле, написав в преамбуле одну из перечисленных ниже команд:

`\scrollmode`	равносильна нажатию `s`
`\nonstopmode`	равносильна нажатию `r`
`\batchmode`	равносильна нажатию `q`

6. Как читать книгу дальше?

Наш обзор основных понятий завершен, и вы уже можете подготовить с помощью LATEX'а несложный текст. Дальнейшее чтение, в зависимости от ваших потребностей, можно построить по-разному: несколько последующих глав почти независимы друг от друга, а внутри каждой из них материал расположен в порядке возрастания трудности.

В главе II подробно рассказано про набор математических формул.

Глава III посвящена набору текста «в малом»: в ней рассказывается, в частности, как задавать в тексте шрифты разных начертаний и размеров, как набирать ударения над буквами и специальные типографские значки наподобие знака параграфа, как делать сноски и т. п.

Глава IV посвящена оформлению текста «в целом»: в ней подробно рассказано про то, какие бывают классы документов и чем они отличаются друг от друга, как устроить разбиение текста на разделы таким образом, чтобы LATEX автоматически создавал заголовки этих разделов и к тому же автоматически их нумеровал, как оформлять титульный лист, как создать оглавление, и тому подобное.

В главе V описаны так называемые псевдорисунки — примитивные картинки, которые можно создавать, не выходя за рамки LaTeX'а.

В главе VI рассказывается о печати таблиц с помощью LaTeX'а.

В главе VII объяснено, как можно повысить эффективность своей работы в LaTeX'е, создавая собственные команды. Первые разделы этой главы можно (а скорее всего, и нужно) читать параллельно с главой II.

Две последние главы предназначены прежде всего для читателей, интересующихся не только набором, но и версткой. В главе VIII рассказывается о таких фундаментальных понятиях TeX'а, как «блоки» и «клей»; заключительная глава IX, предполагающая знание всего предыдущего материала, рассказывает, как изменить стандартный стиль оформления документов, предоставляемый LaTeX'ом, применительно к своим нуждам.

Книга завершается различными приложениями, посвященными более частным вопросам; об их содержании можно узнать из оглавления.

Глава II

Как набирать формулы

О некоторых простейших приемах набора формул уже шла речь в первой главе, но для серьезной работы этого, бесспорно, недостаточно; хочется верить, что после изучения этой главы профессиональному математику (или техническому редактору) не будет страшна никакая, даже самая изощренная, формула.

Некоторые из приемов, описываемых здесь, становятся доступными только после подключения специальных стилевых пакетов. Напомним, что слова наподобие «чтобы сделать то-то и то-то, надо подключить стилевые пакеты `weird` и `queer`» означают, что в преамбулу документа следует включить строчку вида

```
\usepackage{weird,queer}
```

Особенно полезны возможности, предоставляемые пакетами `amssymb` и `amsmath`; рекомендуем подключать их всегда, если в вашем тексте присутствуют сколько-нибудь сложные формулы.

Имея все это в виду, приступим к делу.

1. Таблицы спецзнаков с комментариями

В этом разделе мы перечислим математические знаки, предоставляемые LaTeX'ом. Знаков этих очень много, поэтому разобьем их на несколько групп. Это разбиение делается не только для удобства восприятия: как мы увидим в разд. II.5.4, расстановка интервалов в формулах зависит от того, к какой группе (бинарная операция, бинарное отношение, обыкновенный символ и т. д.) относится математический символ.

1.1. Операции, отношения и просто значки

Начнем с греческих букв. Имя команды, задающей строчную греческую букву, совпадает с английским названием этой буквы (например, буква α задается командой `\alpha`). Исключение составляет буква o (она называется «омикрон»): по начертанию она совпадает с курсивной латинской o, так что специальной команды для нее не предусмотрено, и для ее набора достаточно просто написать `o` в формуле. Некоторые греческие буквы имеют по два варианта начертаний; это также отражено в следующей ниже таблице.

α	`\alpha`	β	`\beta`	γ	`\gamma`
δ	`\delta`	ϵ	`\epsilon`	ε	`\varepsilon`
ζ	`\zeta`	η	`\eta`	θ	`\theta`
ϑ	`\vartheta`	ι	`\iota`	κ	`\kappa`
λ	`\lambda`	μ	`\mu`	ν	`\nu`
ξ	`\xi`	π	`\pi`	ϖ	`\varpi`
ρ	`\rho`	ϱ	`\varrho`	σ	`\sigma`
ς	`\varsigma`	τ	`\tau`	υ	`\upsilon`
ϕ	`\phi`	φ	`\varphi`	χ	`\chi`
ψ	`\psi`	ω	`\omega`		

Имя команды, задающей прописную греческую букву, пишется с прописной буквы (например, буква Ψ задается командой `\Psi`). Некоторые прописные греческие буквы («альфа», например) совпадают по начертанию с латинскими, и для них специальных команд нет — надо просто набрать соответствующую латинскую букву прямым шрифтом (см. с. 62 по поводу того, как это сделать). Не надо использовать греческие буквы Σ и Π из этой таблицы в качестве знаков суммы и произведения: для этих целей есть специальные команды, о которых пойдет речь дальше. Итак, вот прописные греческие буквы, не совпадающие по начертанию с латинскими:

Γ	`\Gamma`	Δ	`\Delta`	Θ	`\Theta`
Λ	`\Lambda`	Ξ	`\Xi`	Π	`\Pi`
Σ	`\Sigma`	Υ	`\Upsilon`	Φ	`\Phi`
Ψ	`\Psi`	Ω	`\Omega`		

Читатель мог заметить, что прописные греческие буквы печатаются, в отличие от строчных, прямым шрифтом. Если вам нужны наклонные прописные греческие буквы (вроде $\mathit{\Sigma}$), прочтите о том, как их получить, в разд. 2.3.

Следующая серия символов — символы, рассматриваемые TEX'ом как символы бинарных операций (наподобие знаков сложения, умножения и т. п.); TEX оставляет в формуле небольшие пробелы по обе стороны этих знаков, кроме случаев, когда есть основания считать, что эти знаки используются не для обозначения операций, а для других целей (если, например, стоят два плюса подряд, то дополнительного пробела между ними не будет). Итак, вот список символов бинарных операций:

$+$	`+`	$-$	`-`	$*$	`*`
$\pm$	`\pm`	$\mp$	`\mp`	$\times$	`\times`
$\div$	`\div`	$\setminus$	`\setminus`	$\cdot$	`\cdot`
$\circ$	`\circ`	$\bullet$	`\bullet`	$\cap$	`\cap`
$\cup$	`\cup`	$\uplus$	`\uplus`	$\sqcap$	`\sqcap`
$\sqcup$	`\sqcup`	$\vee$	`\vee`	$\wedge$	`\wedge`
$\oplus$	`\oplus`	$\ominus$	`\ominus`	$\otimes$	`\otimes`
$\odot$	`\odot`	$\oslash$	`\oslash`	$\triangleleft$	`\triangleleft`
$\triangleright$	`\triangleright`	$\amalg$	`\amalg`	$\diamond$	`\diamond`
$\wr$	`\wr`	$\star$	`\star`	$\dagger$	`\dagger`
$\ddagger$	`\ddagger`	$\bigtriangleup$	`\bigtriangleup`	$\bigcirc$	`\bigcirc`
$\bigtriangledown$	`\bigtriangledown`				

Обозначения для многих из выписанных знаков длинны и сложны. С этим неудобством борются следующим образом: если в вашем тексте часто встречается какое-то длинное обозначение для математического символа, имеет смысл определить для этого символа свою более удобную команду (например, `\btu` вместо `\bigtriangleup`). Как это сделать, рассказано в начале гл. VII; вы можете прочитать это уже сейчас.

В следующей таблице мы собрали символы «бинарных отношений». Вокруг них TEX также оставляет дополнительные пробелы (не такие, как вокруг символов бинарных операций). Вообще говоря, нет смысла много задумываться об этих пробелах, поскольку TEX оформляет математические формулы в достаточно разумном стиле; о тех случаях, когда размер пробелов в математических формулах приходится корректировать вручную, речь пойдет дальше в этой главе.

$<$	`<`	$>$	`>`	$=$	`=`
$:$	`:`	$\le$	`\le`	$\ge$	`\ge`
$\ne$	`\ne`	$\sim$	`\sim`	$\simeq$	`\simeq`
$\approx$	`\approx`	$\cong$	`\cong`	$\equiv$	`\equiv`
$\ll$	`\ll`	$\gg$	`\gg`	$\doteq$	`\doteq`
$\parallel$	`\parallel`	$\perp$	`\perp`	$\in$	`\in`

$\notin$	`\notin`	$\ni$	`\ni`	$\subset$	`\subset`
$\subseteq$	`\subseteq`	$\supset$	`\supset`	$\supseteq$	`\supseteq`
$\succ$	`\succ`	$\prec$	`\prec`	$\succeq$	`\succeq`
$\preceq$	`\preceq`	$\asymp$	`\asymp`	$\sqsubseteq$	`\sqsubseteq`
$\sqsupseteq$	`\sqsupseteq`	$\models$	`\models`	$\vdash$	`\vdash`
$\dashv$	`\dashv`	$\smile$	`\smile`	$\frown$	`\frown`
$\mid$	`\mid`	$\bowtie$	`\bowtie`	$\propto$	`\propto`

Команда `\mid` в этой таблице определяет вертикальную черточку, рассматриваемую как знак бинарного отношения; ее не следует употреблять, если вертикальная черточка употребляется как аналог скобки (например, как знак абсолютной величины). Типичный случай, когда нужна команда `\mid`, — запись определения множеств:

$M = \{\,x \in A \mid x > 0\,\}$ `$M=\{\,x\in A\mid x>0\,\}$`

Если тут написать `|` вместо `\mid`, то пробелы вокруг вертикальной черты будут недостаточны. Команды `\,` нужны, чтобы сделать дополнительные маленькие пробелы возле фигурных скобок (подробнее см. разд. II.5).

Стоит еще отметить, что при записи отображений нужно использовать не двоеточие, а команду `\colon`:

$f\colon X\to Y$ `$f\colon X\to Y$`

Если здесь задать двоеточие непосредственно, то вокруг него получатся слишком большие интервалы.

Если вы подключите стилевой пакет `latexsym`, то вам, кроме того, будут доступны следующие семь символов:

$\lhd$	`\lhd`	$\unlhd$	`\unlhd`	$\rhd$	`\rhd`
$\unrhd$	`\unrhd`	$\sqsubset$	`\sqsubset`	$\sqsupset$	`\sqsupset`
$\Join$	`\Join`				

В следующей таблице собраны стрелки различных видов.

$\to$	`\to`	$\longrightarrow$	`\longrightarrow`	$\Rightarrow$	`\Rightarrow`
$\Longrightarrow$	`\Longrightarrow`	$\hookrightarrow$	`\hookrightarrow`		
$\mapsto$	`\mapsto`	$\longmapsto$	`\longmapsto`		
$\gets$	`\gets`	$\longleftarrow$	`\longleftarrow`	$\Leftarrow$	`\Leftarrow`
$\Longleftarrow$	`\Longleftarrow`	$\hookleftarrow$	`\hookleftarrow`		
$\leftrightarrow$	`\leftrightarrow`	$\longleftrightarrow$	`\longleftrightarrow`		
$\Leftrightarrow$	`\Leftrightarrow`	$\Longleftrightarrow$	`\Longleftrightarrow`		
$\uparrow$	`\uparrow`	$\Uparrow$	`\Uparrow`		

$\downarrow$	`\downarrow`	$\Downarrow$	`\Downarrow`		
$\updownarrow$	`\updownarrow`	$\Updownarrow$	`\Updownarrow`		
$\nearrow$	`\nearrow`	$\searrow$	`\searrow`		
$\swarrow$	`\swarrow`	$\nwarrow$	`\nwarrow`	$\leftharpoondown$	`\leftharpoondown`
$\leftharpoonup$	`\leftharpoonup`	$\rightharpoonup$	`\rightharpoonup`		
$\rightharpoondown$	`\rightharpoondown`	$\rightleftharpoons$	`\rightleftharpoons`		

При подключении стилевого пакета `latexsym` будет также доступна стрелка $\leadsto$ (она задается командой `\leadsto`).

Из привычных российскому читателю символов в вышеприведенных таблицах нет знаков $\geqslant$ и $\leqslant$, выглядящих гораздо лучше, чем $\geq$ и $\leq$; кроме того, греческая буква «каппа» лучше смотрится в виде $\varkappa$, чем в виде κ (`\kappa`). Эти символы становятся доступными, если подключить стилевой пакет `amssymb`. При условии, что это сделано, можно задавать в математических формулах букву $\varkappa$ командой `\varkappa`, а символы $\leqslant$ и $\geqslant$ — командами `\leqslant` и `\geqslant`.

1.2. Операции с пределами и без

В следующей таблице собраны названия функций — команды для воспроизведения названий математических операций наподобие sin, log и т. п., обозначаемых последовательностью букв, набираемых прямым шрифтом. Любую из этих операций можно снабдить верхним и/или нижним индексом (см. пример на с. 33).

log	`\log`	lg	`\lg`	ln	`\ln`
arg	`\arg`	ker	`\ker`	dim	`\dim`
hom	`\hom`	deg	`\deg`	exp	`\exp`
sin	`\sin`	arcsin	`\arcsin`	cos	`\cos`
arccos	`\arccos`	tan	`\tan`	arctan	`\arctan`
cot	`\cot`	sec	`\sec`	csc	`\csc`
sinh	`\sinh`	cosh	`\cosh`	tanh	`\tanh`
coth	`\coth`				

В этой таблице обозначения tan, arctan и т. д. — не что иное, как принятые в англоязычной литературе обозначения для тангенса, арктангенса и т. д. В отечественной литературе, однако же, принято обозначать tg, ctg и т. д. Так как в стандартном комплекте TeX'а или LaTeX'а команд для этого нет, их приходится, при необходимости, определять самому. Это просто: надо подключить пакет `amsmath`, после чего добавить в преамбулу такую строчку:

```
\DeclareMathOperator{\tg}{tg}
```

В первом аргументе команды `\DeclareMathOperator` ставится придуманное вами имя команды (незанятое, естественно), во втором — то, что вы хотите получить на печати. Содержимое второго аргумента будет обработано, как математическая формула, но при этом символы `-` (дефис), `*` и `'` будут иметь такое же значение, как в обычном тексте (это удобно, если вы хотите, чтобы имя вашего нового оператора включало тот же дефис). Разумеется, `\DeclareMathOperator` должно следовать в преамбуле документа после `\usepackage{amsmath}`.

Если не подключать `amsmath`, то собственную функцию типа синуса определить также можно. Для этого достаточно написать в преамбуле документа

```
\newcommand{\tg}{\mathop{\mathrm{tg}}\nolimits}
```

После этого команда `\tg` будет создавать в математической формуле запись tg с правильными пробелами вокруг нее. Другие команды такого типа определяются аналогично, надо только вместо `tg` написать то название функции (скажем, `arctg`), которое должно появиться на печати.

В частности, так приходится делать, чтобы определить команды `\Re` и `\Im` для обозначения вещественной и мнимой части комплексного числа; в LaTeX'е такие команды есть, но на печати они дают не Re и Im, а $\Re$ и $\Im$ (см. с. 53), что не принято в России (да и на Западе не очень принято). При этом, поскольку обозначения `\Re` и `\Im` уже заняты, приходится говорить `\renewcommand` вместо `\newcommand`:

```
\renewcommand{\Im}{\mathop{\mathrm{Im}}\nolimits}
```

Даже при подключенном пакете `amsmath` команда `\DeclareMathOperator` в этом месте не сработает, т. к. уже существующие команды она не переопределяет.

Описанный выше способ определения команд для функций является частным случаем существующей в LaTeX'е конструкции для (пере)определения новых команд (см. гл. VII).

Еще один символ, который принято набирать прямым шрифтом, — это символ mod, используемый в записи «сравнений по модулю». Обычно он употребляется не сам по себе, а в сочетании со знаком $\equiv$ (см. пример ниже); в этом случае для записи сравнения удобна команда `\pmod`, которой пользуются так:

Легко видеть, что $23^{1993} \equiv 1 \pmod{11}$.	`Легко видеть, что $23^{1993}\equiv 1\pmod{11}$.`

Обратите внимание, что скобки вокруг mod 11 получаются автоматически; правая часть сравнения — весь текст, заключенный между `\equiv` и `\pmod`.

Если подключить пакет `amsmath`, то станут доступны команды `\mod` и `\pod`, обозначающие то же понятие, что `\mod`, другими способами:

$a^{p-1}\equiv 1\mod p$	`$a^{p-1}\equiv 1\mod p$`
$a^{p-1}\equiv 1\pod p$	`$a^{p-1}\equiv 1\pod p$`

Иногда символ mod используется и как символ бинарной операции, например, так:

$f_*(x)=f(x)\bmod G$	`$f_*(x)=f(x)\bmod G$`

Как видно из примера, в этом случае надо писать `\bmod`.

Теперь обсудим, как можно было бы получить, скажем, формулу

$$\sum_{i=1}^n n^2=\frac{n(n+1)(2n+1)}{6}$$

с дополнительными элементами над и под знаком операции. В данной формуле эти элементы называются «пределы суммирования», поэтому в TEXнической терминологии записи над и под знаком операции принято называть «пределами» (по-английски limits). В исходном тексте «пределы» обозначаются точно так же, как индексы; имея в виду, что знак суммы генерируется командой `\sum`, заключаем, что вышеназванную формулу можно получить так:

```
$$
  \sum_{i=1}^n n^2=\frac{n(n+1)(2n+1)}{6}
$$
```

В этом примере существенно, что формула была выключной; во внутритекстовой формуле «пределы» печатаются на тех же местах, что и индексы:

Тот факт, что $\sum_{i=1}^n (2n-1)=n^2$, следует из формулы для суммы арифметической прогрессии.	`Тот факт, что $\sum_{i=1}^n (2n-1)=n^2$, следует из формулы для суммы арифметической прогрессии.`

(можно добиться, чтобы пределы и во внутритекстовой формуле были сверху и снизу — см. ниже). Вот список операций, ведущих себя так же, как `\sum`:

$\sum$	`\sum`	$\prod$	`\prod`	$\bigcup$	`\bigcup`
$\bigcap$	`\bigcap`	$\coprod$	`\coprod`	$\bigoplus$	`\bigoplus`
$\bigotimes$	`\bigotimes`	$\bigodot$	`\bigodot`	$\bigvee$	`\bigvee`
$\bigwedge$	`\bigwedge`	$\biguplus$	`\biguplus`	$\bigsqcup$	`\bigsqcup`
$\lim$	`\lim`	$\limsup$	`\limsup`	$\liminf$	`\liminf`
$\max$	`\max`	$\min$	`\min`	$\sup$	`\sup`
$\inf$	`\inf`	$\det$	`\det`	$\Pr$	`\Pr`
$\gcd$	`\gcd`				

Если подключить пакет `amsmath`, то будут доступны еще шесть операций такого типа:

$\varlimsup$	`\varlimsup`	$\varliminf$	`\varliminf`
$\injlim$	`\injlim`	$\projlim$	`\projlim`
$\varinjlim$	`\varinjlim`	$\varprojlim$	`\varprojlim`

Примеры:

$\varlimsup_{n\to\infty} a_n=\inf_n\sup_{m\ge n}a_m$	`$\varlimsup_{n\to\infty}` `a_n=\inf_n\sup_{m\ge n}a_m$`
$\mathcal F_x=\varinjlim_{U\ni x}\mathcal F(U)$	`$\mathcal F_x=` `\varinjlim_{U\ni x}\mathcal F(U)$`

(см. с. 63 по поводу `\mathcal`).

Кроме того, пакет `amsmath` предоставляет возможность определить и собственную команду «с пределами». Для этого надо воспользоваться командой `\DeclareMathOperator*`; синтаксис этой команды такой же, как у команды `\DeclareMathOperator` (см. с. 50), но при «операторе», определенном такой командой, «пределы» будут ставиться так же, как при $\lim$.

Еще одна «математическая операция», для которой требуются «пределы», — это интеграл. В LaTeX'е есть команды `\int` для обычного знака интеграла $\int$ и `\oint` для знака «контурного интеграла» $\oint$; если подключить пакет `amsmath`, то станут доступны также команды `\iint`, `\iiint` и `\iiiint` для двойного, тройного и «четверного» интегралов (если просто написать несколько команд `\int` подряд, то между знаками интеграла получатся слишком большие пробелы).

При этом, для экономии места, пределы интегрирования помещаются не сверху и снизу от знаков интеграла, а по бокам (даже и в выключных формулах):

$$\int_0^1 x^2dx=1/3$$

```
$$
  \int_0^1x^2 dx=1/3
$$
```

Если, тем не менее, необходимо, чтобы пределы интегрирования стояли над и под знаком интеграла, то надо непосредственно после `\int` записать команду `\limits`, а уже после нее — обозначения для пределов интегрирования:

$$\int\limits_0^1 x^2 dx = 1/3$$

```
$$
  \int\limits_0^1 x^2 dx=1/3
$$
```

Тот же прием с командой `\limits` можно применить, если хочется, чтобы во внутритекстовой формуле «пределы» у оператора стояли над и под ним, а не сбоку.

Если, с другой стороны, надо, чтобы в выключной формуле «пределы» у какого-либо оператора стояли не над и под знаком оператора, а сбоку, то после команды для знака оператора надо записать команду `\nolimits`, а уже после нее — обозначения для «пределов»:

$$\prod\nolimits_{i=1}^n i = n!$$

```
$$
  \prod\nolimits_{i=1}^ni=n!
$$
```

1.3. Разное

Мы уже перечислили почти все символы, используемые LaTeX'ом в математических формулах (кроме большого количества тех, доступ к которым открывается при подключении пакета amssymb — см. разд. 1.4 ниже). Остались скобки различных видов (им будет посвящен специальный разд. 2.5), а также ряд значков (среди них есть и часто встречающиеся), не входящих ни в какой из разделов нашей классификации. Они собраны в следующей таблице.

∂	`\partial`	$\triangle$	`\triangle`	$\angle$	`\angle`
∞	`\infty`	$\forall$	`\forall`	$\exists$	`\exists`
$\emptyset$	`\emptyset`	$\neg$	`\neg`	$\aleph$	`\aleph`
$\prime$	`\prime`	$\hbar$	`\hbar`	∇	`\nabla`
$\imath$	`\imath`	$\jmath$	`\jmath`	ℓ	`\ell`
$\surd$	`\surd`	$\flat$	`\flat`	$\sharp$	`\sharp`
$\natural$	`\natural`	$\top$	`\top`	$\bot$	`\bot`
$\wp$	`\wp`	$\Re$	`\Re`	$\Im$	`\Im`
$\backslash$	`\backslash`	$\|$	`|`	$\spadesuit$	`\spadesuit`
$\clubsuit$	`\clubsuit`	$\diamondsuit$	`\diamondsuit`	$\heartsuit$	`\heartsuit`
†	`\dag`	§	`\S`	©	`\copyright`
‡	`\ddag`	¶	`\P`	£	`\pounds`

Последние шесть символов (от † до £) можно использовать не только в формулах, но и в тексте.

Если подключить стилевой пакет `latexsym`, то будут также доступны следующие команды:

$\mho$	`\mho`	$\Box$	`\Box`	$\Diamond$	`\Diamond`

Символ $\emptyset$ (`\emptyset`) — это, конечно, обозначение для пустого множества. В отечественной литературе более принято другое начертание для этого символа: $\varnothing$. Символ пустого множества в этом начертании задается командой `\varnothing`, доступной при подключении стилевого пакета `amssymb`.

Не следует смешивать команды `\parallel` и `\|`. На печати они дадут один и тот же значок $\|$, но с разными пробелами (полиграфист бы сказал «отбивками») вокруг него. Команда `\parallel` нужна для обозначения бинарного отношения «параллельность», в то время как `\|` — это один из видов скобок:

В школьных учебниках геометрии встречаются такие формулы, как $AB\parallel CD$.
В университетских учебниках анализа часто пишут, что $\|A\|=\sup(|Ax|/|x|)$.

```
В школьных учебниках
геометрии встречаются
такие формулы,
как $AB\parallel CD$.

В университетских учебниках
анализа часто пишут,
что $\|A\|=\sup(|Ax|/|x|)$.
```

Символы, обозначаемые командами `\imath` и `\jmath`, нужны для постановки дополнительных значков над буквами i и j (об этом пойдет речь в разд. 2.8).

Команды `\nabla` и `\bigtriangledown` задают разные символы, и их не надо путать. Обратите также внимание на символ, задаваемый командой `\prime`. Это — тот самый штрих, который используется в качестве верхнего индекса, если после символа в формуле поставить знак `'` ; на самом деле записи `x'` и `x^\prime` практически равносильны.

Приведем еще таблицу синонимов. В ней представлены математические символы, которые можно набирать двумя различными способами:

$*$	`*`	или	`\ast`
$\neq$	`\ne`	или	`\neq`
$\leq$	`\le`	или	`\leq`
$\geq$	`\ge`	или	`\geq`
$[$	`[`	или	`\lbrack`
$]$	`]`	или	`\rbrack`
$\{$	`\{`	или	`\lbrace`
$\}$	`\}`	или	`\rbrace`
$\to$	`\to`	или	`\rightarrow`
$\gets$	`\gets`	или	`\leftarrow`
$\ni$	`\ni`	или	`\owns`
$\wedge$	`\wedge`	или	`\land`
$\vee$	`\vee`	или	`\lor`
$\neg$	`\neg`	или	`\lnot`
$\Vert$	`\Vert`	или	`|`

1.4. Символы из пакета amssymb

Математических знаков в стандартном LaTeX'овском наборе очень много, но порой и их не хватает. Если вы подключите пакет amssymb, то сможете воспользоваться дополнительными математическими знаками, разработанными Американским математическим обществом (сокращенно AMS — American Mathematical Society). Этих знаков более полутора сотен; мы опять разобьем их на группы.

Начнем с бинарных операций. Стилевой пакет amssymb дает возможность воспользоваться такими символами, относящимися к этой категории:

$\boxdot$	`\boxdot`	$\boxplus$	`\boxplus`	$\boxtimes$	`\boxtimes`
$\centerdot$	`\centerdot`	$\boxminus$	`\boxminus`	$\veebar$	`\veebar`
$\barwedge$	`\barwedge`	$\doublebarwedge$	`\doublebarwedge`	$\Cup$	`\Cup`
$\Cap$	`\Cap`	$\curlywedge$	`\curlywedge`	$\curlyvee$	`\curlyvee`
$\leftthreetimes$	`\leftthreetimes`	$\rightthreetimes$	`\rightthreetimes`	$\dotplus$	`\dotplus`
$\intercal$	`\intercal`	$\circledcirc$	`\circledcirc`	$\circledast$	`\circledast`
$\circleddash$	`\circleddash`	$\divideontimes$	`\divideontimes`	$\lessdot$	`\lessdot`
$\gtrdot$	`\gtrdot`	$\ltimes$	`\ltimes`	$\rtimes$	`\rtimes`
$\smallsetminus$	`\smallsetminus`				

Следующим по очереди идет огромное количество бинарных отношений:

$\rightleftharpoons$	`\rightleftharpoons`	$\leftrightharpoons$	`\leftrightharpoons`
$\Vdash$	`\Vdash`	$\Vvdash$	`\Vvdash`
$\vDash$	`\vDash`	$\upharpoonright$	`\upharpoonright`
$\downharpoonright$	`\downharpoonright`	$\upharpoonleft$	`\upharpoonleft`
$\downharpoonleft$	`\downharpoonleft`	$\Lsh$	`\Lsh`
$\Rsh$	`\Rsh`	$\circeq$	`\circeq`
$\succsim$	`\succsim`	$\gtrsim$	`\gtrsim`
$\gtrapprox$	`\gtrapprox`	$\multimap$	`\multimap`
$\therefore$	`\therefore`	$\because$	`\because`
$\doteqdot$	`\doteqdot`	$\triangleq$	`\triangleq`
$\precsim$	`\precsim`	$\lesssim$	`\lesssim`
$\lessapprox$	`\lessapprox`	$\eqslantless$	`\eqslantless`
$\eqslantgtr$	`\eqslantgtr`	$\curlyeqprec$	`\curlyeqprec`
$\curlyeqsucc$	`\curlyeqsucc`	$\preccurlyeq$	`\preccurlyeq`
$\leqq$	`\leqq`	$\leqslant$	`\leqslant`
$\lessgtr$	`\lessgtr`	$\risingdotseq$	`\risingdotseq`
$\fallingdotseq$	`\fallingdotseq`	$\succcurlyeq$	`\succcurlyeq`
$\geqq$	`\geqq`	$\geqslant$	`\geqslant`
$\gtrless$	`\gtrless`	$\sqsubset$	`\sqsubset`
$\sqsupset$	`\sqsupset`	$\vartriangleright$	`\vartriangleright`
$\vartriangleleft$	`\vartriangleleft`	$\trianglerighteq$	`\trianglerighteq`
$\trianglelefteq$	`\trianglelefteq`	$\between$	`\between`
$\blacktriangleright$	`\blacktriangleright`	$\blacktriangleleft$	`\blacktriangleleft`
$\vartriangle$	`\vartriangle`	$\eqcirc$	`\eqcirc`
$\lesseqgtr$	`\lesseqgtr`	$\gtreqless$	`\gtreqless`
$\lesseqqgtr$	`\lesseqqgtr`	$\gtreqqless$	`\gtreqqless`
$\varpropto$	`\varpropto`	$\smallsmile$	`\smallsmile`
$\smallfrown$	`\smallfrown`	$\Subset$	`\Subset`
$\Supset$	`\Supset`	$\subseteqq$	`\subseteqq`
$\supseteqq$	`\supseteqq`	$\bumpeq$	`\bumpeq`
$\Bumpeq$	`\Bumpeq`	$\lll$	`\lll`
$\ggg$	`\ggg`	$\pitchfork$	`\pitchfork`
$\backsim$	`\backsim`	$\backsimeq$	`\backsimeq`
$\lvertneqq$	`\lvertneqq`	$\gvertneqq$	`\gvertneqq`
$\lneqq$	`\lneqq`	$\gneqq$	`\gneqq`
$\lneq$	`\lneq`	$\gneq$	`\gneq`
$\precnsim$	`\precnsim`	$\succnsim$	`\succnsim`
$\lnsim$	`\lnsim`	$\gnsim$	`\gnsim`
$\precneqq$	`\precneqq`	$\succneqq$	`\succneqq`
$\precnapprox$	`\precnapprox`	$\succnapprox$	`\succnapprox`
$\lnapprox$	`\lnapprox`	$\gnapprox$	`\gnapprox`
$\varsubsetneq$	`\varsubsetneq`	$\varsupsetneq$	`\varsupsetneq`

$\subsetneqq$	`\subsetneqq`	$\supsetneqq$	`\supsetneqq`
$\varsubsetneqq$	`\varsubsetneqq`	$\varsupsetneqq$	`\varsupsetneqq`
$\subsetneq$	`\subsetneq`	$\supsetneq$	`\supsetneq`
$\eqsim$	`\eqsim`	$\shortmid$	`\shortmid`
$\shortparallel$	`\shortparallel`	$\thicksim$	`\thicksim`
$\thickapprox$	`\thickapprox`	$\approxeq$	`\approxeq`
$\succapprox$	`\succapprox`	$\precapprox$	`\precapprox`
$\backepsilon$	`\backepsilon`		

Несколько символов из этой таблицы нам уже знакомы: в первую очередь это знаки для нестрогих неравенств $\leqslant$ и $\geqslant$ в привычном отечественному читателю начертании, а также знаки $\sqsubset$ и $\sqsupset$, доступ к которым открывается уже при подключении пакета `latexsym`. Символы $\vartriangleleft$, $\trianglelefteq$, $\vartriangleright$ и $\trianglerighteq$, задаваемые командами `\vartriangleleft`, `\trianglelefteq`, `\vartriangleright` и `\trianglerighteq`, также доступны уже при подключении пакета `latexsym`, но там они называются иначе: `\lhd`, `\unlhd`, `\rhd` и `\unrhd` соответственно.

Специальные команды предусмотрены для отрицаний отношений из предыдущей таблицы. В принципе «отрицание» (перечеркнутый символ) можно напечатать, поставив перед этим символом команду `\not` (см. разд. 2.6), но взаимное расположение черты и перечеркиваемого символа при этом не всегда удачно. Поэтому Американское математическое общество выделило для перечеркнутых символов специальные литеры (ради красоты приходится страдать…). Итак:

$\nleq$	`\nleq`	$\ngeq$	`\ngeq`	$\nless$	`\nless`
$\ngtr$	`\ngtr`	$\nprec$	`\nprec`	$\nsucc$	`\nsucc`
$\nleqslant$	`\nleqslant`	$\ngeqslant$	`\ngeqslant`	$\npreceq$	`\npreceq`
$\nsucceq$	`\nsucceq`	$\nleqq$	`\nleqq`	$\ngeqq$	`\ngeqq`
$\nsim$	`\nsim`	$\ncong$	`\ncong`	$\nsubseteqq$	`\nsubseteqq`
$\nsupseteqq$	`\nsupseteqq`	$\nsubseteq$	`\nsubseteq`	$\nsupseteq$	`\nsupseteq`
$\nparallel$	`\nparallel`	$\nmid$	`\nmid`	$\nshortmid$	`\nshortmid`
$\nshortparallel$	`\nshortparallel`	$\nvdash$	`\nvdash`	$\nVdash$	`\nVdash`
$\nvDash$	`\nvDash`	$\nVDash$	`\nVDash`	$\ntrianglerighteq$	`\ntrianglerighteq`
$\ntrianglelefteq$	`\ntrianglelefteq`	$\ntriangleleft$	`\ntriangleleft`	$\ntriangleright$	`\ntriangleright`
$\nleftarrow$	`\nleftarrow`	$\nrightarrow$	`\nrightarrow`	$\nLeftarrow$	`\nLeftarrow`
$\nRightarrow$	`\nRightarrow`	$\nLeftrightarrow$	`\nLeftrightarrow`	$\nleftrightarrow$	`\nleftrightarrow`

В следующей таблице мы собрали всевозможные стрелки (с точки зрения TEX'а, стрелки — это тоже знаки бинарных отношений, но математики, как правило, так не считают).

$\circlearrowright$	`\circlearrowright`	$\circlearrowleft$	`\circlearrowleft`
$\twoheadrightarrow$	`\twoheadrightarrow`	$\twoheadleftarrow$	`\twoheadleftarrow`

$\leftleftarrows$	`\leftleftarrows`	$\rightrightarrows$	`\rightrightarrows`
$\upuparrows$	`\upuparrows`	$\downdownarrows$	`\downdownarrows`
$\rightarrowtail$	`\rightarrowtail`	$\leftarrowtail$	`\leftarrowtail`
$\leftrightarrows$	`\leftrightarrows`	$\rightleftarrows$	`\rightleftarrows`
$\rightsquigarrow$	`\rightsquigarrow`	$\leftrightsquigarrow$	`\leftrightsquigarrow`
$\looparrowleft$	`\looparrowleft`	$\looparrowright$	`\looparrowright`
$\Rrightarrow$	`\Rrightarrow`	$\Lleftarrow$	`\Lleftarrow`
$\nleftarrow$	`\nleftarrow`	$\nrightarrow$	`\nrightarrow`
$\nLeftarrow$	`\nLeftarrow`	$\nRightarrow$	`\nRightarrow`
$\nLeftrightarrow$	`\nLeftrightarrow`	$\nleftrightarrow$	`\nleftrightarrow`
$\curvearrowleft$	`\curvearrowleft`	$\curvearrowright$	`\curvearrowright`

Стрелка $\rightsquigarrow$ доступна и при использовании пакета `latexsym`, но там она называется `\leadsto`.

В приведенной выше таблице присутствует команда `\rightleftharpoons`, входящая в основной набор LaTeX'а; ниже можно найти еще несколько аналогичных примеров. Такое дублирование — не прихоть TeXнологов из AMS: в базовом LaTeX'е символы, задаваемые этими дублирующимися командами, собирались из отдельных кусочков, вследствие чего они не меняли должным образом размеры при помещении в индексы. В пакете `amssymb` те же команды отсылают к специальным литерам, входящим в шрифты AMS, в результате чего символы $\rightleftharpoons$ (`\rightleftharpoons`) или, скажем, $\hbar$ (`\hbar`) правильно ведут себя и в индексах.

Теперь перечислим обыкновенные символы, доступ к которым открывается при подключении пакета `amssymb`:

$\square$	`\square`	$\blacksquare$	`\blacksquare`
$\lozenge$	`\lozenge`	$\blacklozenge$	`\blacklozenge`
$\backprime$	`\backprime`	$\bigstar$	`\bigstar`
$\blacktriangledown$	`\blacktriangledown`	$\blacktriangle$	`\blacktriangle`
$\triangledown$	`\triangledown`	$\angle$	`\angle`
$\measuredangle$	`\measuredangle`	$\sphericalangle$	`\sphericalangle`
$\circledS$	`\circledS`	$\complement$	`\complement`
$\diagup$	`\diagup`	$\diagdown$	`\diagdown`
$\varnothing$	`\varnothing`	$\nexists$	`\nexists`
$\Finv$	`\Finv`	$\Game$	`\Game`
$\mho$	`\mho`	$\eth$	`\eth`
$\beth$	`\beth`	$\gimel$	`\gimel`
$\daleth$	`\daleth`	$\digamma$	`\digamma`
$\varkappa$	`\varkappa`	$\Bbbk$	`\Bbbk`
$\hslash$	`\hslash`	$\hbar$	`\hbar`

Из этого набора нам знакомы символ $\mho$, доступ к которому открывает уже пакет `latexsym`, а также греческая буква $\varkappa$ и обозначение для пустого множества $\varnothing$. Символы $\square$ и $\lozenge$ также доступны уже при подключении пакета `latexsym`, но там они называются `\Box` и `\Diamond` соответственно.

Команды `\yen`, `\checkmark`, `\circledR` и `\maltese` можно использовать не только в формулах, но и в тексте (кстати, ¥ — это обозначение для японской иены).

У некоторых из символов, определенных в пакете `amsfonts`, тоже есть синонимы. Вот их список:

$\dashrightarrow$	`\dasharrow`	или	`\dashrightarrow`
$\doteqdot$	`\Doteq`	или	`\doteqdot`
$\Cup$	`\Cup`	или	`\doublecup`
$\Cap$	`\Cap`	или	`\doublecap`
$\lll$	`\lll`	или	`\llless`
$\ggg$	`\ggg`	или	`\gggtr`

Стоит еще отметить, что некоторые из определенных выше символов становятся доступными уже при использовании пакета `amsfonts` вместо `amssymb`. Их перечень таков:

¥	`\yen`	✓	`\checkmark`
®	`\circledR`	✠	`\maltese`
$\dashrightarrow$	`\dasharrow`	$\dashleftarrow$	`\dashleftarrow`
$\sqsubset$	`\sqsubset`	$\sqsupset$	`\sqsupset`
$\vartriangleleft$	`\vartriangleleft`	$\vartriangleright$	`\vartriangleright`
$\trianglelefteq$	`\trianglelefteq`	$\trianglerighteq$	`\trianglerighteq`
$\square$	`\square`	$\lozenge$	`\lozenge`
$\rightsquigarrow$	`\rightsquigarrow`	$\rightleftharpoons$	`\rightleftharpoons`

Поскольку пакет `amssymb` довольно громоздок, стоит иметь в виду возможность иногда обойтись более скромным `amsfonts`.

1.5. Какие еще есть символы

В наших таблицах собрано более четырехсот математических символов, не считая операций типа log или sin. Тем не менее, для набора формул этого мало. Во-первых, в формулах встречаются скобки разнообразных начертаний (и размеров). О них речь пойдет в разд. 2.5. Во-вторых, часто бывает нужно, чтобы обозначения переменных в формуле печатались не курсивом, как это делается в TeX'е по умолчанию, а другим шрифтом. Как этого добиться, рассказано в разд. 2.3. Если вам и этого не хватает, попробуйте поискать недостающие шрифты в Интернете (в приложении Ж рассказано, где именно искать).

2. Важные мелочи

2.1. Нумерация формул

В математических текстах обычно приходится для удобства ссылок нумеровать формулы; LaTeX позволяет организовать эту нумерацию таким образом, чтобы номера формул и ссылки на них генерировались автоматически (см. разд. I.2.11). Нумеровать таким образом можно только *выключные* формулы. Делается это так.

Выключная формула, которую вы нумеруете, должна быть оформлена как окружение `equation` (знаков `$$` быть не должно!). Каждая такая формула на печати автоматически получит номер. Чтобы на него можно было ссылаться, надо формулу пометить: в любом месте между `\begin{equation}` и `\end{equation}` поставить команду `\label`, и после этого команда `\ref` будет генерировать номер формулы (см. с. 27; напомним, что может понадобиться повторный запуск LaTeX'а). Поясним сказанное примером:

Как известно,

$$7 \times 9 = 63. \tag{1}$$

....

Из формулы (1) следует, что $63/9 = 7$.

```
Как известно,
\begin{equation}
\label{trivial}
7\times9=63.
\end{equation}
........................
Из формулы~(\ref{trivial})
следует, что $63/9=7$.
```

Знак `~` мы поставили, чтобы номер формулы и слово «формулы» не попали на разные строки (см. с. 103). Обратите внимание, что скобки вокруг номера формулы, сгенерированного командой `\ref`, автоматически *не* ставятся. Если вы подключили пакет `amsmath`, то можете воспользоваться командой `\eqref`, единственным отличием которой от `\ref` является то, что она автоматически ставит скобки вокруг номера формулы.

Можно также использовать команду `\pageref` вместо `\ref` — тогда на печати получится не номер формулы, а номер страницы, на которую попала эта формула.

То, как именно выглядит на печати номер формулы, зависит от класса документа (см. с. 20, 155): например, в классе `article` (статья) формулы имеют сплошную нумерацию, а в классе `book` (книга) нумерация формул начинается заново в каждой главе, и номер, скажем, формулы 5 из главы 3, генерируемый окружением `equation`, имеет вид (3.5). В главе, посвященной модификации стандартных классов, мы расскажем, как можно самостоятельно менять вид номеров формул.

Кроме того, вы можете вообще не пользоваться автоматической генерацией номеров формул, а ставить их вручную. Чтобы номер выглядел при этом красиво, удобно воспользоваться TeX'овской командой `\eqno`. Следующий пример показывает, как это делать:

Простое тождество

$$7 \times 9 = 63 \qquad (3.2)$$

известно каждому школьнику.

```
Простое тождество
$$
  7\times9=63\eqno (3.2)
$$
известно каждому школьнику.
```

Выключная формула, нумеруемая с помощью команды `\eqno`, должна быть оформлена с помощью знаков `$$`; номером формулы будет служить весь текст, заключенный между `\eqno` и закрывающими формулу `$$`; этот текст обрабатывается TeX'ом так же, как математические формулы (стало быть, пробелы игнорируются, буквы печатаются «математическим курсивом», и т. п.). Можно также вместо `\eqno` сказать `\leqno`, тогда ваш номер формулы будет не справа, а слева.

Никаких автоматических ссылок на формулу, генерируемую командой `\eqno` или `\leqno`, TeX не создает, и в этом случае за корректность ссылок отвечаете только вы.

2.2. Переносы в формулах

При необходимости TeX может перенести часть внутритекстовой формулы на другую строчку. Такие переносы возможны после знаков «бинарных отношений», наподобие знака равенства[1] (см. с. 47) или «бинарных операций», наподобие знаков сложения или умножения (см. с. 47), причем последний знак в строке, вопреки российской традиции, не дублируется в начале следующей. Чтобы избежать этих переносов, можно воспользоваться тем обстоятельством, что TeX не разрывает при переносе часть формулы, заключенную в фигурные скобки. В частности, можно заключить в фигурные скобки всю формулу, в которой произошел нежелательный перенос, от открывающего ее знака доллара до закрывающего: после этого можно быть уверенным, что переноса этой формулы ни при каких обстоятельствах не произойдет.

Вышеописанный способ борьбы с неудачными переносами в формулах имеет один недостаток: при этом затрудняется верстка абзацев и возрастает вероятность появления неприятных сообщений «`Overfull \hbox`» (см. разд. III.6).

Более гибкий способ борьбы с переносами в формулах — записать в преамбуле файла строку

[1]Стрелки также рассматриваются TeX'ом как бинарные отношения.

```
\binoppenalty=10000
```

и/или строку

```
\relpenalty=10000
```

Первая из этих строк запретит все разрывы строк после знаков бинарных операций, а вторая — после знаков бинарных отношений, и при этом помех верстке абзаца будет меньше, чем при заключении всей формулы в фигурные скобки.

Для любознательных поясним, что `\binoppenalty` и `\relpenalty` — параметры (TEX'овские), значением которых может быть целое число (см. с. 25 по поводу TEX'овских параметров). Эти параметры определяют степень нежелательности разрыва строки после символов бинарной операции и бинарного отношения соответственно (чем больше значение соответствующего параметра, тем менее желателен разрыв строки). По умолчанию значение `\binoppenalty` равно 700, а значение `\relpenalty` равно 500. Можно присвоить им в преамбуле бóльшие значения, тогда вероятность разрывов уменьшится. Значение 10000 означает абсолютный запрет.

При заключении всей формулы в фигурные скобки верстка абзацев затрудняется, поскольку TEX лишается возможности варьировать в ней интервалы между символами для выравнивания строк (см. разд. III.6).

Наконец, существует способ дублировать знаки операций, который мы приведем безо всяких пояснений. Включив

```
\newcommand*{\hm}[1]{#1\nobreak\discretionary{}%
            {\hbox{$\mathsurround=0pt #1$}}{}}
```

в преамбулу, можно написать`$a\hm+b\hm+c\hm+d$`, при этом в формуле $a+b+$ $+c+d$ при переносе знак $+$ будет продублирован.

Выключные формулы, в отличие от внутритекстовых, TEX *никогда* не переносит. Если выключная формула не помещается в строку, то при трансляции вы получите сообщение «`Overfull \hbox`» (в разд. III.6 подробно рассказано, в каких еще ситуациях выдается такое сообщение), и вам придется разбить формулу на строки вручную. Как это делать, мы объясним в разд. 4.2.

2.3. Смена шрифтов в формуле

По умолчанию все латинские буквы в формулах набираются курсивом. Что делать, если вам нужен другой шрифт?

В первой главе мы приводили примеры смены шрифтов в тексте с помощью команд наподобие `\bfseries` или `\itshape`. В формулах, однако же, для этих целей надо использовать другие средства.

Пусть, например, вам нужна буква $\mathbf{P}$, набранная прямым жирным шрифтом. Тогда надо воспользоваться командой `\mathbf`:

$\mathbf P^n$	`$\mathbf P^n$`

Если буква **P** (в таком начертании) встречается в формулах часто, разумно определить для нее сокращенное обозначение. Чтобы узнать, как это делается, посмотрите начало главы VII.

Вот полный список начертаний символов в формулах, которые можно получить без подключения дополнительных стилевых пакетов:

$\mathbf x+y$	`$\mathbf x+y$`
$\mathrm x+y$	`$\mathrm x+y$`
$\mathtt x+y$	`$\mathtt x+y$`
$\mathsf x+y$	`$\mathsf x+y$`
$\mathcal T_X$	`$\mathcal T_X$`
$\mathit\Gamma+y$	`$\mathit\Gamma+y$`

Команду `\mathcal`, вызывающую «каллиграфический» шрифт, можно применять только к прописным латинским буквам.

Команда `\mathit` дает одну из возможностей (и не лучшую, надо сказать) получить прописные греческие буквы в наклонном начертании. Дело в том, что если у вас подключен пакет `amsmath`, то команда `\mathit` работать откажется (по крайней мере, в некоторых версиях этого пакета). В этом случае для печати в формуле прописных греческих букв в наклонном начертании применяются специальные команды, в которых перед названием буквы стоит `var`:

$\varGamma$	`\varGamma`	$\varDelta$	`\varDelta`	$\varTheta$	`\varTheta`
$\varLambda$	`\varLambda`	$\varXi$	`\varXi`	$\varPi$	`\varPi`
$\varSigma$	`\varSigma`	$\varUpsilon$	`\varUpsilon`	$\varPhi$	`\varPhi`
$\varPsi$	`\varPsi`	$\varOmega$	`\varOmega`		

Как мог заметить читатель, команды наподобие `\mathrm` действуют только на непосредственно следующую букву. Если нужно, чтобы другим шрифтом была напечатана не одна буква, а несколько, надо все эти буквы взять в фигурные скобки:

Множество особенностей многообразия X обозначается X_{sing}.	`Множество особенностей многообразия $X$ обозначается $X_{\mathrm{sing}}$.`

Все сказанное означает, что команда `\mathrm` и ей подобные принимают один обязательный аргумент — фрагмент формулы, который надо напечатать другим шрифтом. На первый взгляд, это противоречит сказанному на с. 23: ведь обязательный аргумент должен быть в фигурных скобках, а в конструкциях вроде `\mathbf x` никаких фигурных скобок нет. Дело в том, что, в дополнение к сказанному на с. 23, действует еще одно правило: если после имени команды, принимающей обязательный аргумент, следует не открывающая фигурная скобка, а буква, то в качестве аргумента будет воспринята именно эта буква. Так что можно было бы писать и `\mathbf{x}` вместо `\mathbf x`, но так обычно не делают, чтобы не нажимать лишний раз на клавиши. Ср. с. 107.

Если подключить стилевой пакет `amsfonts` или `amssymb`, то в математических формулах можно использовать еще два шрифта: ажурный ($\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}$,...) и готический ($\mathfrak{S}$, $\mathfrak{p}$, $\mathfrak{g}$,...). Ажурным шрифтом можно печатать только прописные буквы; он задается командой `\mathbb` (как и в случае с остальными командами, описываемыми в этом разделе, ажурным шрифтом печатается буква, следующая непосредственно после команды `\mathbb`; если надо напечатать этим шрифтом несколько букв, их следует взять в фигурные скобки). Готический шрифт задается командой `\mathfrak`; она также действует только на непосредственно следующую букву (или на несколько букв, если они взяты в фигурные скобки):

Алгебра $\mathfrak{sl}_2(\mathbb{C})$ играет особую роль в теории представлений.	`Алгебра $\mathfrak{sl}_2(\mathbb C)$ играет особую роль в теории представлений.`

Существует также стилевой пакет `eufrak`, при подключении которого становится доступным готический шрифт (с командой `\mathfrak`), но не ажурный.

Наконец, есть возможность использовать в формулах вариант рукописного шрифта, в котором буквы имеют более изысканные очертания:

$\mathscr{A}\quad \mathscr{B}\quad \mathscr{C}\quad \mathscr{D}\quad \mathscr{E}\quad \mathscr{F}\ldots$

Для этого надо подключить стилевой пакет `euscript`; команда, задающая этот шрифт, называется `\EuScript`.

Теперь, когда вы знаете, как печатать символы в формулах прямым шрифтом, может возникнуть искушение восполнить отсутствие в стандартном комплекте LaTeX'а команды, дающей функцию tg, путем набора чего-нибудь вроде `\mathrm{tg}x`. Так делать, однако, не надо, поскольку при этом пробелы будут неправильными:

В формуле $\mathrm{tg} x$ буква x слишком близка к знаку тангенса. А вот в формуле $\sin x$ пробелы правильные.	`В формуле $\mathrm{tg} x$ буква $x$ слишком близка к знаку тангенса. А вот в формуле $\sin x$ пробелы правильные.`

Правильно действовать так, как рекомендуется на с. 50. Если вам хочется узнать, почему все так получается, прочтите разд. 5.4.

Если вы хотите включить в формулу какой-либо текст, то одной команды `\mathrm` для этого также недостаточно: любой текст, заключенный между знаками доллара, пусть даже он набирается прямым шрифтом, TeX рассматривает как часть математической формулы, и в соответствии с этим игнорирует те пробелы, которые ставите вы, и расставляет пробелы по собственным правилам:

$\sqrt{x^3} = x для всех x.$	`$$ \sqrt{x^3}=x \mathrm{для всех} x. $$`

Как правильно вставить текст в формулу, описано в разд. 2.4.

Остается только отметить, что ажурный и готический шрифты, о которых шла речь сейчас, можно использовать только в формулах, и набирать с их помощью обычный текст невозможно (так же, как невозможно набирать греческий текст с помощью команд `\alpha`, `\beta` и т. д.).

Кроме описанного выше (и рекомендуемого нами) способа переключения шрифтов в формулах, в LaTeX'е пока сохраняется (ради совместимости с LaTeX'ом 2.09) еще один способ, с которым можно ознакомиться из приведенной ниже таблицы.

Правильный способ:	Устаревший способ:	Получается:
`\mathrm x+y`	`{\rm x}+y`	$\mathrm{x}+y$
`\mathbf x+y`	`{\bf x}+y`	$\mathbf{x}+y$
`\mathsf x+y`	`{\sf x}+y`	$\mathsf{x}+y$
`\mathtt x+y`	`{\tt x}+y`	$\mathtt{x}+y$
`\mathcal T_X`	`{\cal T}_X`	$\mathcal{T}_X$

2.4. Включение текста в формулы

В математическую формулу можно включить фрагмент обычного текста с помощью LaTeX'овской команды `\mbox`. В следующем примере продемонстрировано, как это можно сделать; в нем используется еще команда `\qquad`, делающая в тексте или формуле пробел размером `2em` (см. с. 26 по поводу единиц длины, применяемых TeX'ом); подробнее по поводу команд, создающих пробелы в формулах, см. разд. 5.1; по поводу команд, создающих пробелы в тексте, см. разд. III.3.3.

$$\sqrt{x^3} = x \qquad \mbox{для всех } x.$$

```
$$
  \sqrt{x^3}=x\qquad
  \mbox{для всех }x.
$$
```

Аргумент команды `\mbox` обрабатывается TeX'ом как обычный текст: пробелы не игнорируются, слова набираются не математическим курсивом, а тем же шрифтом, который был текущим перед началом формулы (у нас это был обычный прямой шрифт; если вы хотите, чтобы шрифт был другой, можно внутри аргумента команды `\mbox` дать команду смены шрифта в тексте). Весь текст, являющийся аргументом команды `\mbox`, будет напечатан в одну строку. В приведенном примере мы оставили пробел перед закрывающей фигурной скобкой, чтобы обеспечить пробел между текстом и формулой (фрагмент текста, созданный командой `\mbox`, рассматривается TEX'ом как одна большая буква; пробел в формуле между «буквой», содержащей текст из `\mbox`, и буквой x будет недостаточен). Команда `\qquad` была использована по аналогичной причине.

На самом деле можно было бы написать даже так:

```
$$
  \sqrt{x^3}=x\qquad\mbox{для всех $x$.}
$$
```

Аргумент команды `\mbox` рассматривается как текст, но этот текст вполне может, в свою очередь, содержать формулы!

При включении текста в формулы с помощью команды `\mbox` важно иметь в виду вот что. Как читатель, видимо, уже заметил, в математических формулах верхние и нижние индексы, числитель и знаменатель дробей, созданных с помощью команды `\frac`, и тому подобные фрагменты набираются более мелким шрифтом, чем остальная часть формулы. Однако в тексте, созданном с помощью `\mbox`, размер шрифта не изменится, в какую бы часть формулы этот текст не попал. Если вас это не устраивает, подключите стилевой пакет `amsmath` и воспользуйтесь командой `\text` вместо `\mbox`: тогда включенный в формулу текст будет правильно менять размеры в степенях и индексах, а если аргумент команды `\text` сам, в свою очередь, содержит формулу, то размеры символов в этой формуле будут выбраны более правильно, чем при использовании `\mbox`.

Вне математических формул команду `\text` использовать нельзя.

Команда `\mbox` еще встретится нам в следующей главе, когда речь пойдет о предотвращении переносов в словах; более подробно мы ее рассмотрим в главе VIII.

2.5. Скобки переменного размера

Если заключенный в скобки фрагмент формулы занимает много места по вертикали (за счет дробей, степеней и тому подобного), то и сами скобки должны быть большего размера, чем обычные. В TEX'е на этот случай предусмотрен механизм автоматического выбора размера скобок. Пользуются им так.

В формуле

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

скобки обычного размера вокруг $1+\frac{1}{n}$ смотрелись бы плохо; поэтому при ее наборе надо поставить команду `\left` перед открывающей скобкой и команду `\right` перед закрывающей:

```
$$
  e=\lim_{n\to\infty}
     \left(1+\frac{1}{n}\right)^n
$$
```

Если перед одной скобкой стоит `\left`, а перед другой скобкой стоит `\right`, то на печати размер этих скобок будет соответствовать максимальной высоте фрагмента формулы, заключенного между `\left` и `\right`.

Конструкция с `\left` и `\right` применима не только к круглым скобкам. В следующей таблице перечислены скобки и некоторые другие символы, которые с помощью `\left` и `\right` автоматически принимают нужный размер. TEXнический термин для таких символов — ограничители (по-английски delimiters).

<table>
<tr><td>$($</td><td><code>(</code></td><td>$)$</td><td><code>)</code></td><td>$[$</td><td><code>[</code></td></tr>
<tr><td>$]$</td><td><code>]</code></td><td>$\{$</td><td><code>\{</code></td><td>$\}$</td><td><code>\}</code></td></tr>
<tr><td>$\lfloor$</td><td><code>\lfloor</code></td><td>$\rfloor$</td><td><code>\rfloor</code></td><td>$\lceil$</td><td><code>\lceil</code></td></tr>
<tr><td>$\rceil$</td><td><code>\rceil</code></td><td>$\langle$</td><td><code>\langle</code></td><td>$\rangle$</td><td><code>\rangle</code></td></tr>
<tr><td>$\vert$</td><td><code>|</code></td><td>$\Vert$</td><td><code>\|</code></td><td>$/$</td><td><code>/</code></td></tr>
<tr><td>$\backslash$</td><td><code>\backslash</code></td><td></td><td></td><td></td><td></td></tr>
</table>

Вместо `\left\langle` можно писать `\left<` , и аналогичным образом вместо `\right\rangle` можно писать `\right>` (однако же `<` нельзя писать вместо `\langle`!). Кроме знаков, перечисленных в этой таблице, менять свои размеры под действием `\left` и `\right` могут и вертикальные стрелки из таблицы на с. 48.

Если подключить стилевой пакет `amssymb`, то станут доступны еще две пары ограничителей:

$\ulcorner$	`\ulcorner`	$\urcorner$	`\urcorner`	$\llcorner$	`\llcorner`	$\lrcorner$	`\lrcorner`

Вместе с каждой командой `\left` в формуле должна присутствовать соответствующая ей команда `\right`, в противном случае TEX выдаст сообщение об ошибке. Вместе с тем TEX вовсе не требует, чтобы «ограничители» (например, скобки) при командах `\left` и `\right` были расположены сколько-нибудь осмысленно с математической точки зрения: вы вполне можете написать что-нибудь вроде `\left(...\right]` , или даже, вопреки смыслу слов left и right, `\left)...\right(` — за правильность своих формул отвечаете только вы, и $\TeX$ тут вам не помощник.

Вместо «ограничителя» после команды `\left` или `\right` можно поставить точку. На месте этой точки ничего не напечатается, а другой «ограничитель» будет необходимого размера. Вот два примера того, как можно использовать этот прием. Во-первых, таким способом можно создать косую дробную черту увеличенного размера (символ `/` также является «ограничителем» — см. таблицу):

$$M(f)=\left.\left(\int\limits_a^b f(x)\,dx\right)\right/(b-a)$$

```
$$
M(f)=\left.\left(
     \int\limits_a^b
       f(x)\,dx
     \right)
     \right/(b-a)
$$
```

В этом примере используется пока неизвестная вам команда `\,` , создающая дополнительный маленький пробел между $f(x)$ и dx — это один из немногих случаев, когда $\TeX$ не может автоматически создать требуемые пробелы, и ему надо помочь. Подробнее о таких вещах речь пойдет ниже, в разд. 5.1. Другой пример использования ограничителя без пары таков:

$$\int\limits_a^b\frac12(1+x)^{-3/2}dx=\left.-\frac{1}{\sqrt{1+x}}\right|_a^b$$

```
$$
  \int\limits_a^b\frac12
       (1+x)^{-3/2}dx=
  \left.-\frac{1}{\sqrt{1+x}}
       \right|_a^b
$$
```

Здесь, кстати, мы не поставили `\,` перед `dx`, поскольку необходимое свободное место возникает за счет показателя степени.

Наконец, важный пример использования ограничителей без пары — использование их для набора систем уравнений, о чем пойдет речь в разд. 4.2.

До сих пор у нас речь шла только о том, что размеры ограничителей выбираются автоматически с помощью команд `\left` и `\right`; бывают, однако, ситуации, когда такой автоматический выбор размера приводит к неудовлетворительным результатам или даже вообще невозможен. Вот, например, ситуация, когда `\left` и `\right` не срабатывают:

$\left| |x+1|-|x-1|\right|$

```
$\left| |x+1|-|x-1|\right|$
```

Для удобочитаемости этого выражения хотелось бы, чтобы внешние знаки модуля были повыше, чем внутренние, но этого не получается: поскольку в формуле выступающих элементов нет, то и команды `\left` и `\right` не считают нужным увеличить ограничители, в которые формула заключена.

А иногда бывает так, что автоматически получающиеся ограничители слишком велики. В следующем примере совсем не обязательно, чтобы скобки охватывали и пределы суммирования, что получается при использовании `\left` и `\right`:

$$\left(\sum_{k=1}^n x^k\right)^2$$

```
$$
\left(
\sum_{k=1}^n x^k
\right)^2
$$
```

Во всех этих случаях имеет смысл указать размер ограничителя явно. Для этого предусмотрены TeX'овские команды `\bigl`, `\Bigl`, `\biggl` и `\Biggl` для левых ограничителей и `\bigr`, `\Bigr`, `\biggr` и `\Biggr` для правых ограничителей. Мы перечислили эти команды в порядке возрастания размера создаваемого ими ограничителя. В частности, для модулей можно было бы написать так:

$\bigl| |x+1|-|x-1|\bigr|$

```
$\bigl| |x+1|-|x-1|\bigr|$
```

Пример со знаком суммы кому-то мог бы понравиться больше, если бы мы написали так:

$$\Bigl(\sum_{k=1}^n x^k\Bigr)^2$$

```
$$
\Bigl(
\sum_{k=1}^n x^k
\Bigr)^2
$$
```

Команды, явно указывающие размер ограничителей, не обязаны, в отличие от команд `\left` и `\right`, появляться парами: можно написать `\biggl(` и при этом никак не упомянуть о парной скобке.

К сожалению, команды для явного указания размера ограничителя имеют одну неприятную особенность: если «основной шрифт» документа крупнее, чем кегль 10 (иными словами, если указаны классовые опции `11pt` или `12pt` — см. с. 155 и ниже), то может случиться так, что скобка, размер которой задан, например, командой `\bigl`, имеет точно такой же размер, как и скобка «в чистом виде». Чтобы избежать этой неприятности, надо подключить пакет `amsmath`: тогда команды наподобие `\bigl` будут работать корректно.

2.6. Перечеркнутые символы

Чтобы получить в математической формуле изображение перечеркнутого символа, надо перед командой, генерирующей этот символ, поставить команду `\not`. Пример:

Множество $\{x \mid x \not\ni x\}$ существовать не может. В этом состоит парадокс Рассела.

```
Множество $\{x\mid x\not\ni x\}$
существовать не может. В этом
состоит парадокс Рассела.
```

Кстати, для получения знака $\notin$ лучше не писать `\not\in`, а воспользоваться командой `\notin` — при этом знак получится более красивым. Если подключен пакет `amssymb`, то можно также вместо команды `\not` пользоваться готовыми командами для перечеркнутых символов (с. 57).

2.7. Формула в рамочке

Очень важную формулу хочется взять в рамку. Если подключить пакет `amsmath`, то этого можно добиться с помощью команды `\boxed`:

$$\boxed{\iint_{\mathbb R^2} e^{-(x^2+y^2)}\,dx\,dy=\pi}$$

```
$$\boxed{
\iint_{\mathbb R^2}
e^{-(x^2+y^2)}\,dx\,dy=\pi
}$$
```

В этом примере мы подразумевали, что подключен еще пакет `amsfonts` или `amssymb`.

2.8. Надстрочные знаки

Часто требуется поставить дополнительный значок над буквой или фрагментом формулы: черточку, «крышку», и т. п. В TeX'е для этих целей есть специальные команды.

Во-первых, можно поставить горизонтальную черту над любым фрагментом формулы с помощью команды `\overline`, как в следующем примере:

Часто используется обозначение

$$\overline{a_na_{n-1}\ldots a_1a_0}=10^na_n+\ldots+a_0.$$

Особенно часто так пишут в научно-популярных книгах.

```
Часто используется
обозначение
$$
  \overline{a_na_{n-1}\ldots
  a_1a_0}=10^na_n+\ldots+a_0.
$$
Особенно часто так пишут в
научно-популярных книгах.
```

Для постановки других значков над буквами в формулах предусмотрены команды, перечисленные в следующей таблице, в которой, для примера, эти значки ставятся над буквой a:

`\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`	$\vec a$

Между прочим, команда `\bar` ставит не совсем такую же черточку, как `\overline`.

Если поставить значок над буквой i или j, так, чтобы сохранилась и точка над буквой, то это будет некрасиво. Поэтому значки следует ставить не прямо над этими буквами, а над символами $\imath$ и $\jmath$ (см. таблицу на с. 53):

Писать $\tilde i$ некрасиво; лучше писать так: $\tilde\imath$.

```
Писать $\tilde i$ некрасиво;
лучше писать так: $\tilde\imath$.
```

Надстрочные знаки, перечисленные в таблице, можно ставить только над одиночными буквами: если сказать `\hat{a+b}`, то получится некрасивая формула $\hat{a+b}$; TeX предоставляет возможность поставить «крышку» подходящего размера над целым фрагментом формулы с помощью команды `\widehat`:

Тождество $\widehat{f*g}=\hat f\cdot\hat g$ означает, что преобразование Фурье переводит свертку в произведение.

```
Тождество $\widehat{f*g}=
\hat f\cdot\hat g$ означает,
что преобразование Фурье
переводит свертку в произведение.
```

Аналогичным образом можно поставить «волну» над фрагментом формулы с помощью команды `\widetilde`. В отличие от горизонтальной черты, генерируемой командой `\overline`, знаки, генерируемые командами `\widehat` и `\widetilde`, не могут быть сколь угодно широкими (максимально возможная ширина — в примере выше).

Кроме того, существует команда `\overrightarrow`, предназначенная для постановки стрелки над формулой:

Рассмотрим вектор $\overrightarrow{AB}$.

```
Рассмотрим вектор
$\overrightarrow{AB}$.
```

Аналогичная ей команда `\overleftarrow` ставит над формулой стрелку, направленную влево, а не вправо.

Остальные команды для постановки акцентов в формулах не имеют «широких» вариантов.

Формулы типа `\hat{\hat A}`, в которых акцент ставится над буквой, уже имеющей акцент, могут выглядеть неудачно. Если вам нужны такие «двойные акценты», подключите пакет `amsmath` и пользуйтесь командами `\Hat`, `\Check`, `\Tilde`, `\Acute`, `\Grave`, `\Dot`, `\Ddot`, `\Breve`, `\Bar` и `\Vec`:

Правильно $\hat{\hat{Z}}$, а не $\hat{\hat{Z}}$.

```
Правильно $\Hat{\Hat Z}$, а не $\hat{\hat Z}$.
```

(Для одиночных акцентов эти команды применять тоже можно.)

Впрочем, в последних версиях пакета `amsmath` команды типа `\hat` исправлены и действуют так же, как их аналоги с большой буквы.

Отметим, что TeX позволяет ставить надстрочные знаки над буквами не только в математической формуле, но и в обычном тексте (в этом случае такие знаки обычно называют «диакритическими»), но команды для постановки этих знаков совершенно другие. Об этом — на с. 106.

2.9. Альтернативные обозначения для математических формул

Наряду со стандартными TeX'овскими обозначениями для математических формул, LaTeX предоставляет альтернативные обозначения. Именно, внутритекстовую формулу, которая в стандартных обозначениях ограничивается одним знаком доллара в начале и одним в конце, можно вместо этого заключить в знаки `\(` (в начале) и `\)` (в конце):

$2 \times 2 = 4$

```
\(2\times2=4\)
```

Другой вариант обозначений для внутритекстовой формулы, предоставляемый LaTeX'ом, — написать `\begin{math}` в начале формулы и `\end{math}` в конце (иными словами, внутритекстовая формула может быть оформлена как окружение с именем `math`).

Выключную формулу LaTeX позволяет окружить с обеих сторон не только парами знаков доллара, как предусмотрено стандартом, но знаками `\[` (в начале) и `\]` (в конце). Кроме того, можно оформить выключную формулу как окружение с именем `displaymath`. В одном и том же файле можно использовать как стандартные, так и LaTeX'овские обозначения для формул.

Эти альтернативные обозначения полностью эквивалентны стандартным TeX'овским (со знаками доллара), за одним важным исключением: если выключные формулы обозначаются LaTeX'овскими, а не TeX'овскими обозначениями, то можно сделать так, что выключные формулы будут не центрированы, а прижаты влево (см. с. 159).

3. Набор матриц

Сначала мы объясним, как набирать матрицы при подключенном пакете amsmath (что во всех отношениях лучше и удобнее), а в конце этого раздела расскажем, для полноты картины, о тех средствах набора матриц, которые доступны в «чистом» LaTeX'е (без подключения дополнительных стилевых пакетов).

Итак, предположим, что пакет amsmath подключен. Тогда для набора матриц, заключенных в круглые скобки, стоит воспользоваться окружением pmatrix. Вот как оно работает:

$$\begin{pmatrix}
a_{11}-\lambda & a_{12}&a_{13}\\
a_{21}& a_{22}-\lambda &a_{23}\\
a_{31}& a_{32}&a_{33}-\lambda
\end{pmatrix}$$

```
$$\begin{pmatrix}
a_{11}-\lambda & a_{12}&a_{13}\\
a_{21}& a_{22}-\lambda &a_{23}\\
a_{31}& a_{32}&a_{33}-\lambda
\end{pmatrix}$$
```

Строки матрицы разделяются с помощью команды `\\` (последнюю строку заканчивать командой `\\` не надо), а элементы внутри одной строки, относящиеся к разным столбцам, отделяются друг от друга с помощью символа &. Текст, соответствующий на печати одной строке матрицы, не обязан укладываться в одну строку TeX'овского файла; в одной строке TeX'овского файла можно поместить текст, соответствующий на печати нескольким строкам матрицы. Короче говоря, в окружении matrix также действует TeX'овский принцип «конец строки равносилен пробелу».

Прямоугольные таблицы из формул бывают заключены не только в круглые скобки; соответственно, определены окружения bmatrix, vmatrix и Vmatrix, отличающиеся от pmatrix только тем, что вместо круглых скобок таблица заключена соответственно в квадратные скобки [ ], вертикальные черточки | | и удвоенные вертикальные черточки ‖ ‖. Есть также окружение matrix, которое дает на печати только прямоугольную таблицу, без всяких скобок. Комбинируя окружение matrix с парой ограничителей, можно получить матрицу со скобками более экзотического вида.

Если вам нужны матрицы с более чем десятью столбцами, нужно изменить максимальное количество столбцов, написав в преамбуле что-нибудь вроде следующего:

```
\setcounter{MaxMatrixCols}{20}
```

(после этого максимальное число столбцов в матрице станет равно двадцати; на TeX'ническом языке это действие называется «присваивание нового значения счетчику MaxMatrixCols»; см. главу VII). Можно также дать эту команду не в преамбуле, а в начале той выключной формулы, в которую входит ваша матрица; тогда разрешение увеличить число столбцов будет действительно только для матриц, входящих в эту выключную формулу.

Вот как можно набрать с помощью окружения `matrix` треугольник Паскаля:

$$
\begin{matrix}
&&&& 1 && 1\\
&&& 1 && 2 && 1\\
&& 1 && 3 && 3 && 1\\
& 1 && 4 && 6 && 4 && 1\\
1 && 5 && 10 && 10 && 5 && 1
\end{matrix}
$$

Исходный текст для него выглядит так:

```
$$
\setcounter{MaxMatrixCols}{20}
  \begin{matrix}
   &&&& 1 && 1\\
   &&& 1 && 2 && 1\\
   && 1 && 3 && 3 && 1\\
   & 1 && 4 && 6 && 4 && 1\\
   1 && 5 && 10 && 10 && 5 && 1
  \end{matrix}
$$
```

(заметим кстати, что в этом примере пустые элементы таблицы в конце строки опущены, так что число символов `&` в разных строках таблицы разное). Если бы мы не увеличивали `MaxMatrixCols`, то последняя строка вызвала бы сообщение об ошибке.

Чтобы получить в матрице горизонтальный ряд точек, простирающийся на несколько столбцов, используется команда `\hdotsfor`; ее обязательный аргумент — количество столбцов, занятых точками. В приведенном ниже примере обратите внимание на расстановку знаков `&` в строках, содержащих `\hdotsfor`:

$$\begin{vmatrix}
0 & 0&\hdotsfor{2} &a_1\\
1 & 0&\hdotsfor{2} &a_2\\
\hdotsfor{5}\\
\hdotsfor{2} &1 &0 &a_{n-1}\\
0 & \hdotsfor{2} &1 &a_n
\end{vmatrix}$$

```
$$\begin{vmatrix}
0 & 0&\hdotsfor{2} &a_1\\
1 & 0&\hdotsfor{2} &a_2\\
\hdotsfor{5}\\
\hdotsfor{2} &1 &0 &a_{n-1}\\
0 & \hdotsfor{2} &1 &a_n
\end{vmatrix}$$
```

Можно также регулировать густоту точек, получаемых при помощи команды `\hdotsfor`: в необязательном аргументе (он ставится перед обязательным) можно указать десятичную дробь — «коэффициент разреживания». Если сказать `\hdotsfor[1.5]{5}` вместо `\hdotsfor{5}`, то точки будут идти в полтора раза реже.

Наряду с горизонтальными рядами точек, в матрицах приходится использовать вертикальные и диагональные многоточия. Для их набора используются команды `\vdots` и `\ddots`:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n}\\ a_{21} & a_{22} & \ldots & a_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}$$

```
$$
 \begin{pmatrix}
 a_{11}& a_{12} &\ldots & a_{1n}\\
 a_{21}& a_{22} &\ldots & a_{2n}\\
 \vdots& \vdots &\ddots & \vdots\\
 a_{n1}& a_{n2} &\ldots & a_{nn}
 \end{pmatrix}
$$
```

Команды `\vdots` и `\ddots` можно использовать не только в матрицах, но и в любом месте в математических формулах.

Наряду с матрицами, используемыми в выключных формулах, иногда приходится поместить небольшую матрицу и в формулу внутритекстовую. Естественно, и размеры символов, и интервалы между ними в такой матрице должны быть поскромнее. Для таких целей предназначено окружение `smallmatrix` (оно также становится доступным при подключении пакета `amsmath`). Вот пример его использования:

$[X,Y]=\bigl(\begin{smallmatrix} 1 & 0\\0 & -1 \end{smallmatrix}\bigr)$

```
$[X,Y]=\bigl(\begin{smallmatrix}
1 & 0\\0 & -1
\end{smallmatrix}\bigr)$
```

Как вы могли заметить, скобки вокруг такой маленькой матрицы приходится ставить самостоятельно. Никаких вариантов с готовыми скобками у окружения `smallmatrix` нет.

Теперь, как мы и обещали, сообщим, какие возможности для набора матриц остаются, если не подключать дополнительных пакетов. В этом случае необходимо пользоваться LaTeX'овским окружением `array`. Вот как получить этими средствами пример со с. 73:

$$\left(\begin{array}{ccc} a_{11}-\lambda & a_{12} & a_{13}\\ a_{21} & a_{22}-\lambda & a_{23}\\ a_{31} & a_{32} & a_{33}-\lambda \end{array}\right)$$

```
$$\left(\begin{array}{ccc}
a_{11}-\lambda & a_{12}&a_{13}\\
a_{21}& a_{22}-\lambda &a_{23}\\
a_{31}& a_{32}&a_{33}-\lambda
\end{array}\right)
$$
```

По сравнению с тем, что дает `pmatrix`, отличия следующие:

1) Скобки вокруг матрицы, набираемой с помощью окружения `array`, всегда надо задавать самостоятельно.

2) После `\begin{array}`, открывающего окружение, должна следовать (в фигурных скобках, поскольку это аргумент окружения `array`) так называемая *преамбула* матрицы, описывающая, сколько и каких столбцов должно быть в матрице. В нашем случае преамбула представляет собой три буквы `ccc`. Это значит, что в матрице 3 столбца (по букве на столбец), и что содержимое каждого из этих столбцов должно быть расположено по центру столбца (`c` — от слова «centered»). (Кроме `c`, в преамбуле может стоять буква `l`, означающая, что соответствующий столбец будет выровнен по левому краю (left), или `r`, означающая, что столбец будет выровнен по правому краю (right).)

В остальном синтаксис такой же, как для окружения `pmatrix` и его аналогов. Команды `\ldots`, `\vdots` и `\ddots` по-прежнему можно использовать, а вот `\hdotsfor` — увы, нет. Аналога `MaxMatrixCols` для окружения `array` также нет (поскольку преамбула и так определяет точное число столбцов). Окружение `smallmatrix` в «чистом» LaTeX'е (без подключения дополнительных пакетов) также не предусмотрено.

4. Одно над другим

В этом разделе речь пойдет о тех случаях, когда в формуле необходимо поместить один символ над другим. В разд. 1.2 уже шла речь о частном случае этой проблемы: постановке «пределов» у знака суммы, интеграла или чего-нибудь еще в этом роде. Сейчас мы рассмотрим общий случай.

4.1. Простейшие случаи

Для начала рассмотрим такие возможности расположения одной части формулы над другой:

1) Верхняя часть формулы расположена немного выше строки, нижняя — немного ниже (как в дроби, создаваемой командой `\frac`, но, возможно, без дробной черты).

2) Нижняя часть формулы расположена вровень с остальным текстом, верхняя — над ним.

3) Над или под фрагментом формулы проведена горизонтальная фигурная скобка, а над или под этой скобкой расположен другой фрагмент формулы.

Разберем эти варианты последовательно.

Начнем с одного дополнения по поводу описанной в первой главе команды `\frac`, задающей дроби. Если дробь, заданная с помощью команды `\frac`, встречается во внутритекстовой формуле, то ее числитель и знаменатель печатаются довольно мелким шрифтом, что не всегда приемлемо. Чтобы этого избежать, можно, подключив пакет `amsmath`, воспользоваться командой `\dfrac`: тогда шрифт будет более крупным. Если дробь во внутритекстовой формуле входит в показатель степени или индекс, то иногда имеет смысл задавать ее с помощью команды `\tfrac` (опять-таки чтобы шрифт был не слишком мелким; эта команда также доступна при подключении `amsmath`). Вот примеры:

$\frac23$ и $\dfrac23$	`$\frac23$ и $\dfrac23$`
$2^{\frac35}$ и $2^{\tfrac35}$	`$2^{\frac35}$ и $2^{\tfrac35}$`

Теперь о том, как расположить части формулы «так же, как в дроби», но без дробной черты. Для этого есть два (к сожалению, взаимоисключающих) способа: с подключением пакета `amsmath` и без этого пакета.

Если у вас подключен пакет `amsmath`, можно добиться требуемого эффекта с помощью ограничителей и окружения `smallmatrix`:

Раньше вместо Γ^k_{ij} писали $\left\{\begin{smallmatrix}ij\\k\end{smallmatrix}\right\}$.

```
Раньше вместо~$\Gamma^k_{ij}$
писали~$\left\{
\begin{smallmatrix}ij\\k
\end{smallmatrix}\right\}$.
```

Конечно, если таких формул у вас в тексте много, пользоваться столь длинными обозначениями немыслимо: нужно на базе `smallmatrix` разработать сокращенное обозначение (прочтите в главе VII, как определять «макросы с параметрами»).

Для наиболее часто встречающегося случая «биномиальных коэффициентов», когда ограничителями являются обычные круглые скобки, в пакете `amsmath` предусмотрена специальная команда `\binom`, работающая аналогично `\frac`:

$\binom{12}7=792$	`$\binom{12}7=792$`

У команды `\binom` есть также аналоги `\dbinom` и `\tbinom`, относящиеся к ней так же, как `\dfrac` и `\tfrac` относятся к `\frac`.

В пакете `amsmath` предусмотрена также конструкция «обобщенной дроби», предназначенная для создания команд, аналогичных `\frac` и `\binom`. По определению, обобщенная дробь — это фрагмент формулы, устроенный так: левый ограничитель, затем дробь (толщина дробной черты может быть произвольной, в том числе нулевой), затем правый ограничитель. Напомним, что ограничители — это скобки и им подобные символы, способные автоматически менять размер (с. 67); в обобщенной дроби ограничители могут и отсутствовать (так что обычная дробь — действительно частный случай обобщенной). Для набора обобщенной дроби предусмотрена команда `\genfrac` с шестью аргументами. Чтобы понять, как она работает, посмотрим на пример:

Формула $\genfrac{(}{]}{1pt}{0}{x}{y-z}$ лишена всякого смысла.

```
Формула
$\genfrac{(}{]}{1pt}{0}{x}{y-z}$
лишена всякого смысла.
```

Первый и второй аргументы команды `\genfrac` — это левый и правый ограничители соответственно; третий аргумент — толщина дробной черты (если толщина нулевая, то дробная черта не печатается); четвертый аргумент содержит указания по поводу размера шрифта для числителя и знаменателя: если оставить его пустым, написав просто `{}` вместо `{0}`, то TEX выберет размер самостоятельно; цифра `0` означает, что размер символов будет таким же, как при пользовании командой `\dfrac` (в разд. 5.2 вы узнаете, что в TEX'нической терминологии это называется `displaystyle`), цифра 1 — размер, как при пользовании командой `\tfrac` (он же `textstyle`), цифры 2 и 3 задают еще более мелкие размеры; наконец, пятый и шестой аргументы — это собственно числитель и знаменатель.

Если оставить третий аргумент пустым, написав просто `{}` вместо фигурных скобок, в которых записана толщина, то будет выбрана толщина дробной черты по умолчанию (она равна 0.4 пункта). Если оставить первый и второй аргумент пустыми, то ограничителей не будет (если, однако, левый ограничитель указан, то должен быть указан и правый). Например, `\dfrac{x}{y}` — это то же самое, что

```
\genfrac{}{}{}{0}{x}{y}
```

В частности, наш пример с символом Кристоффеля можно записать как

```
$\genfrac{\{}{\}}{0pt}{}{ij}{k}$
```

Конечно, команда `\genfrac` хороша не сама по себе, а как сырье для определения макросов, приспособленных к вашим конкретным нуждам.

Теперь о том, как быть, если вы не подключаете пакет `amsmath`.

В этом случае удобно воспользоваться TEX'овской командой `\atop`:

Раньше вместо Γ^k_{ij} писали $\left\{{ij\atop k}\right\}$.

```
Раньше вместо~$\Gamma^k_{ij}$
писали~$\left\{ij\atop k\right\}$.
```

В данном случае мы воспользовались еще командами `\left` и `\right` для постановки фигурных скобок необходимого размера.

Для биномиальных коэффициентов есть TEX'овская команда `\choose`:

$$\binom{n}{k}=\frac{n!}{k!(n-k)!}$$

```
$$
{n\choose k}=\frac{n!}{k!(n-k)!}
$$
```

Обратите внимание на фигурные скобки, в которые мы заключили выражение `n\choose k`: команда `\choose` помещает сверху часть формулы от открывающей фигурной скобки до `\choose`, а снизу — часть формулы от `\choose` до закрывающей фигурной скобки. Если бы этих фигурных скобок не было, вниз пошла бы и вся дробь $\frac{n!}{k!(n-k)!}$ вместе со знаком равенства.

Команда `\atop` определяет, что пойдет вверх, а что — вниз, по тем же правилам, что и `\choose`. В примере выше с `\atop` мы обошлись без фигурных скобок, поскольку в математической формуле их функцию исполняют также команды `\left` и `\right`.

При подключенном пакете `amsmath` командами `\atop` и `\choose` пользоваться нельзя.

Интересный случай использования дробей — так называемые «цепные дроби»:

$$\frac{7}{25}=\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3}}}}$$

Наивная попытка набрать эту формулу выглядит так:

$$\frac{7}{25}=\frac{1}{3+\frac{1}{1+\frac{1}{1+\frac{1}{3}}}}$$

```
$$
  \frac{7}{25}=
  \frac{1}{
   3+\frac{1}{
    1+\frac{1}{
     1+\frac{1}{3}}}}
$$
```

Результат смотрится не лучшим образом. В разд. 5 объясняется, почему все получилось так плохо и как исправить это положение «вручную», но на практике лучше всего подключить пакет `amsmath` и сделать так:

$$\frac{7}{25}=\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3}}}}$$

```
$$\frac{7}{25}=
  \cfrac{1}{
   3+\cfrac{1}{
    1+\cfrac{1}{
     1+\cfrac{1}{3}}}}
$$
```

Если вы хотите, чтоб какой-то из числителей в цепной дроби был не центрирован, а выключен влево или вправо, надо вместо `\cfrac` сказать `\cfrac[l]` или `\cfrac[r]` соответственно.

Еще один случай, когда надо напечатать две формулы одинакового размера одну под другой, встречается, когда выражение для индексов суммирования занимает несколько строчек. В этом случае надо, подключив пакет `amsmath`, воспользоваться командой `\substack`:

$$\sum_{\substack{i\in[0;n]\\ j\in[0;m]}} a_{ij}$$

```
$$
 \sum_{\substack{i\in[0;n]\\
        j\in[0;m]}} a_{ij}
$$
```

В единственном аргументе команды `\substack` записываются формулы, которые должны быть под знаком суммы (или произведения, или любой другой «операции с пределами»); строчки разделяются знаком `\\` (как и в окружениях, предназначенных для набора матриц).

Рассмотрим случай, когда нижняя часть формулы должна остаться на уровне строки. Чтобы добиться этого эффекта, используется LaTeX'овская команда `\stackrel`. У этой команды два аргумента: первый — то, что будет над строкой, второй — то, что останется в строке:

$A\stackrel{f}{\longrightarrow}B$

```
$A\stackrel{f}{\longrightarrow}B$
```

Если текст, который надо написать над стрелкой, длинный, прием со `\stackrel` даст неудовлетворительные результаты. В этом случае надо, подключив пакет `amsmath`, воспользоваться командами `\xleftarrow` и `\xrightarrow`, специально предназначенными для нанесения надписей над и под стрелками. В обязательном аргументе этих команд ставится надпись над стрелкой, в необязательном — под стрелкой (необязательный аргумент, если он есть, ставится перед обязательным). Если надпись длинная, размер стрелки автоматически увеличивается:

$$A\xleftarrow[z]{f}B \xrightarrow{f+g-h}C$$

```
$$ A\xleftarrow[z]{f}B
\xrightarrow{f+g-h}C $$
```

Наконец, чтобы нарисовать горизонтальную фигурную скобку под выражением (а под этой скобкой еще, возможно, и сделать подпись), надо воспользоваться командой `\underbrace`. Аргумент этой команды — тот фрагмент формулы, под которым надо провести скобку; подпись под скобкой, если она нужна, оформляется как нижний индекс. Например, такая формула

$$\underbrace{1+3+5+7+\ldots+(2n-1)}_{n\ \text{слагаемых}}=n^2$$

получается следующим образом:

```
$$
  \underbrace{1+3+5+7+
  \ldots+(2n-1)}_{\mbox{$n$ слагаемых}}=n^2
$$
```

Если у вас подключен пакет amsmath, разумно воспользоваться командой \text вместо \mbox.

Горизонтальная фигурная скобка над фрагментом формулы генерируется командой \overbrace, надпись над ней оформляется как верхний индекс. В одной формуле могут присутствовать горизонтальные фигурные скобки как над, так и под фрагментом формулы:

$$\overbrace{\underbrace{a+b+\ldots+z}_{26}+1+\ldots+10}^{36}$$

```
$$
  \overbrace{\underbrace{
              a+b+\ldots+z
                  }_{26}+1+
   \ldots+10}^{36}
$$
```

В нашем примере нижняя горизонтальная скобка была расположена целиком внутри верхней горизонтальной скобки. Можно сделать и так, чтобы верхняя и нижняя горизонтальные скобки не содержали одна другую, а перекрывались, но для этого нужны дополнительные хитрости (с. 93).

4.2. Многострочные выключные формулы

Программа TeX никогда не делает автоматических переносов в выключных формулах, поэтому, если ваша формула не умещается в строку, необходимо разбить ее на отдельные строки самостоятельно. Первое, что приходит в голову начинающим, — это оформить каждую из этих строк как отдельную выключную формулу с помощью `$$...$$` и записать эти выключные формулы подряд. При этом расстояние по вертикали между двумя строками получается слишком большим, так что на глаз они не воспринимаются как части одной формулы. В этом разделе мы описываем, как грамотно организовать такое разбиение.

Как и в случае с матрицами, наиболее удобные (и рекомендуемые нами) средства открываются, если подключить пакет amsmath; с их описания мы и начнем, а в конце опишем скромные средства набора многострочных формул, доступные без подключения дополнительных пакетов.

Итак, пусть вы подключили amsmath. Тогда самое простое средство для набора многострочных выключных формул — это окружение multline:

$$\begin{multline}1+2+3+4+\ldots\\+46+47+48+\ldots\\+99+100=5050\tag{2}\end{multline}$$

```
\begin{multline}
1+2+3+4+\ldots\\
+46+47+48+\ldots\\
+99+100=5050
\end{multline}
```

Первая из строк печатается выключенной влево, последняя — выключенной вправо, остальные строки центрируются. Подобно окружению `equation`, окружение `multline` *не* должно быть заключено в знаки `$$`. Как вы могли заметить, формула, оформленная в виде окружения `multline`, автоматически нумеруется. Чтобы этой нумерации не было, надо воспользоваться «вариантом со звездочкой» — окружением `multline*`.

На самом деле первая и последняя строки печатаются не вплотную к полям, а с отступом, равным `\multlinegap`. Значение этого параметра можно изменить обычным образом, написав в преамбуле что-нибудь вроде

```
\multlinegap=.5in
```

(см. с. 25).

Чтобы какая-то из средних строк была не центрирована, а выключена влево, надо воспользоваться командой `\shoveleft`, написав, скажем,

```
\shoveleft{+46+47+48+\ldots}\\
```

вместо `+46+47+48+\ldots\\`. Для выключки вправо аналогичным образом используется команда `\shoveright`.

Когда несколько выключных формул идут подряд, можно не оформлять каждую из них с помощью `$$` или окружения `equation`, но воспользоваться окружением `gather`:

$$\begin{gather}2\times 2=4\tag{3}\\9\times 9=81\tag{4}\end{gather}$$

```
\begin{gather}
2\times 2=4\\
9\times 9=81
\end{gather}
```

При использовании `gather` формулы также не должны быть заключены в символы `$$`. Каждая из формул, собранных в `gather`, автоматически нумеруется. Чтобы на пронумерованную таким образом формулу можно было сослаться (а иначе зачем нумеровать?), надо ее пометить, поставив перед `\\` команду `\label` (см. примеры меток и ссылок в разд. 2.1; подробности — в разд. IV.9 ниже).

Если какую-то из них нумеровать не надо, следует поставить непосредственно перед `\\` команду `\notag`. Если вы не хотите нумеровать ни одну из формул, можно воспользоваться «вариантом со звездочкой» — окружением `gather*`.

При разбиении выключной формулы на части нередко бывает желательно расположить строки одна под другой так, чтобы они были определенным образом выровнены. Для достижения такого эффекта удобно воспользоваться окружением `split`:

$$\begin{split}1999&=1000+900+{}\\ &+90+9\end{split} \tag{5}$$

```
\begin{equation}
\begin{split}
1999&=1000+900+{}\\
  &+90+9
\end{split}
\end{equation}
```

Разбиение формулы на строки по-прежнему задается с помощью `\\`, а знак `&` стоит перед символами, по которым производится выравнивание. По TEXническим причинам формулу, разбитую на строки помощью `split`, нельзя задавать с помощью знаков `$$` (почему мы и воспользовались в примере окружением `equation`). С другой стороны, из-за `equation` наша формула получила номер. Если нумерация вам не нужна, можно либо написать `\notag` перед `\end{equation}`, либо воспользоваться окружением `equation*`, которое формулы не нумерует.

Формулы, разбитые на части с помощью `split`, можно использовать также внутри окружений `gather` или `align` (о последнем речь пойдет ниже), со звездочками или без.

Нередко возникает необходимость напечатать один или несколько выровненных столбцов формул. Для этих целей предназначено окружение `align`:

$$\begin{aligned} 7\times 9&=63 & 63:9&=7 \qquad (6)\\ 9\times 10&=90 & 90:10&=9 \qquad (7)\end{aligned}$$

```
\begin{align}
7\times 9& =63 & 63:9& =7\\
9\times 10& =90 & 90:10& =9
\end{align}
```

При выравнивании формул по знаку равенства (или другого бинарного отношения), как это обычно и делается, знак `&` ставится *перед* знаком равенства. В нашем примере второй знак `&` в строке отделяет первый столбец формул от второго, по третьему знаку `&` идет выравнивание во втором столбце, четвертый `&`, если бы он был, отделял бы второй столбец от третьего, и т. д. По-прежнему не нужны знаки `$$`, каждая строка уравнений автоматически получает номер, который можно подавить, написав `\notag` перед `\\`, и по-прежнему есть вариант со звездочкой `align*`, который формулы не нумерует.

При грамотном применении окружения `align` в строке должно стоять нечетное число знаков `&`. Именно, если у нас n столбцов с уравнениями, то имеется $n-1$ знаков `&`, отделяющих друг от друга столбцы, плюс еще n знаков — по одному на каждый столбец, а всего $(n-1)+n=2n-1$.

Полезное применение `align` возникает, когда идущие подряд выключные формулы содержат текстовые комментарии. Желательно, чтобы эти комментарии были выровнены. Вот как можно этого добиться с помощью `align`:

$$\begin{aligned}3\cdot 5+7\cdot 5&=(3+7)\cdot5 &&\text{(ясно)}\\ &=50&&\text{(очевидно)}\end{aligned}$$

```
\begin{align*}
3\cdot 5+7\cdot 5&=(3+7)
\cdot5 &&\text{(ясно)}\\
 &=50&&\text{(очевидно)}
\end{align*}
```

Обратите внимание на два амперсенда, отделяющие комментарий от формул (см. выше текст мелким шрифтом). Нелишне также отметить, что, как и в случае с окружениями `multline` и `gather`, формулы, задаваемые с помощью `align`, нельзя оформлять с помощью знаков доллара.

Не всегда удобно включать комментарии к выкладкам прямо в формулы. Иногда хочется, чтобы какой-то из комментариев шел в отдельной строке. Команда `\intertext` позволяет сделать это так, чтобы выравнивание не нарушилось:

$$\begin{aligned}3\cdot 5+7\cdot 5&=(3+7)\cdot5 &&\text{(ясно)}\\ &=50&&\text{(очевидно),}\end{aligned}$$

откуда

$$15+35=50$$

```
\begin{align*}
3\cdot 5+7\cdot 5&=(3+7)
\cdot5 &&\text{(ясно)}\\
&=50&&\text{(очевидно),}\\
\intertext{откуда}
15+35 &=50
\end{align*}
```

Наряду с окружением `align`, дающим сразу целую выключную формулу, есть окружение `aligned`, которое можно использовать в качестве составной части большей формулы. Вот как можно с помощью этого окружения задать систему уравнений:

$$\left\{\begin{aligned}x^2+y^2&=7\\ x+y&=3.\end{aligned}\right.$$

```
$$
  \left\{
   \begin{aligned}
    x^2+y^2&=7\\
      x+y & = 3.\\
   \end{aligned}
  \right.
$$
```

Для создания фигурной скобки, охватывающей всю систему, мы воспользовались командами `\left` и `\right`, причем при команде `\right` стоит «пустой ограничитель» — точка (см. разд. 2.5).

Наконец, еще один тип многострочных выключных формул возникает, когда выражение в правой части равенства должно выглядеть по-разному в разных случаях. На этот случай в пакете `amsmath` предусмотрено окружение `cases`. Продемонстрируем его работу сразу на примере:

$$|x| = \begin{cases} x, & \text{если } x>0; \\ 0, & \text{если } x=0; \\ -x, & \text{если } x<0. \end{cases}$$

```
$$
|x|=\begin{cases}
x,&\text{если $x>0$;}\\
0,&\text{если $x=0$;}\\
-x,&\text{если $x<0$.}
\end{cases}
$$
```

Разработчики пакета `amsmath` рекомендуют внутри окружения `cases` использовать команду `\lbrace` вместо `\{`.

Теперь, когда вы ознакомились с возможностями набора многострочных формул с помощью пакета `amsmath`, расскажем и о том, что можно сделать в этом направлении без подключения дополнительных стилевых пакетов.

Системы уравнений можно набирать с помощью окружения `array` таким образом:

$$\left\{ \begin{array}{rcl} x^2+y^2 & = & 7 \\ x+y & = & 3. \end{array} \right.$$

```
$$
  \left\{
   \begin{array}{rcl}
    x^2+y^2&=&7\\
     x+y & = &3.\\
   \end{array}
  \right.
$$
```

Мы отвели по одному столбцу на левую часть каждого уравнения, на знак равенства и на правую часть. При этом мы попросили, чтоб левые части уравнений были выровнены по правому краю (отсюда `r` в преамбуле), правые части выровнены по левому краю (`l` в преамбуле), а знак равенства располагался по центру своей колонки (поэтому вторая буква в преамбуле — буква `c`).

Можно заметить, что пробелы (отбивки) до и после знака равенства получаются больше, чем это допускается типографскими правилами (и чем получается при использовании окружения `aligned` из пакета `amsmath`). К сожалению, бороться с этим трудно; проще раздобыть комплект, в который входит пакет `amsmath`.

Если необходимо, чтобы отдельные уравнения в системе были пронумерованы, можно воспользоваться окружением `eqnarray`. Оно работает так же, как окружение `array` с преамбулой `rcl` в вышеприведенном примере, но при этом у каждого уравнения автоматически печатается его номер (подобно тому, как автоматически печатается номер у выключной формулы, созданной с помощью окружения `equation` — см. разд. 2.1). Если пометить какое-либо уравнение с помощью команды `\label`, то в дальнейшем можно на него ссылаться с помощью команды `\ref` или `\pageref`. Пример:

$$\begin{array}{rcll} 2\times 3 & = & 6 & (8)\\ 2+3 & = & 5 & (9) \end{array}$$

На с. 86 приведено глупое уравнение 9.

```
\begin{eqnarray}
2\times3&=&6\\
2+3&=&5\label{silly}
\end{eqnarray}
На с.~\pageref{silly}
приведено глупое
уравнение~\ref{silly}.
```

Обратите внимание, что фигурной скобки, охватывающей систему уравнений, окружение `eqnarray` не создает. В этом примере символ `~` между «с.» и `\pageref` поставлен, чтобы слово «с.» и номер страницы не попали на разные строки (см. с. 103); для аналогичных целей мы использовали этот символ и вторично.

При использовании окружения `eqnarray` *не* надо писать знаки `$$` (подобно тому, как не надо их писать при пользовании окружением `equation`).

Если вы хотите нумеровать не все уравнения, надо уравнения, которые вы нумеровать не будете, пометить командой `\nonumber` (непосредственно перед `\\`):

$$\begin{array}{rcll} \int_{-\infty}^{\infty} e^{-x^2}dx & = & \sqrt{\pi} & \\ \sqrt{576} & = & 24 & (10) \end{array}$$

```
\begin{eqnarray}
\int_{-\infty}^\infty
e^{-x^2}dx & = &
\sqrt{\pi}\nonumber\\
\sqrt{576} & = & 24
\end{eqnarray}
```

Наконец, если вы вообще не хотите нумеровать уравнения, то можно воспользоваться «вариантом со звездочкой» — окружением `eqnarray*`.

Окружение `array` можно использовать не только в выключных, но и во внутритекстовых формулах, хотя результат при этом обычно выглядит некрасиво. Окружения `eqnarray` и `eqnarray*` создают только выключные формулы.

Чтобы разбить выключную формулу на несколько выровненных частей, также можно воспользоваться окружением `eqnarray` или `eqnarray*`:

$$\begin{array}{rcl} x^{20} & = & (x-1)^{20}+20(x-1)^{19}+\ldots+\\ & & {}+20(x-1)+1 \end{array}$$

```
\begin{eqnarray*}
x^{20}&=&(x-1)^{20}+
  20(x-1)^{19}+\ldots+\\
  &&{}+20(x-1)+1
\end{eqnarray*}
```

Обратите внимание, что перед первым знаком `+` во второй строке формулы мы поставили пару из открывающей и закрывающей фигурных скобок; это сделано для того, чтобы на печати знак $+$ не подошел слишком близко к первому символу второй строки, что в сочетании со увеличенными отбивками вокруг знака равенства было бы уже слишком (можете поставить эксперимент самостоятельно). Природа описанного эффекта объясняется ниже в разд. 5; он частично учтен в пакете `amsmath` (к сожалению, разные версии этого пакета могут давать разные результаты).

4.3. Набор коммутативных диаграмм

Чтобы набирать в LaTeX'е «коммутативные диаграммы», необходимо подключить стилевой пакет `amscd`. Пусть это сделано. Тогда коммутативная диаграмма оформляется в виде окружения `CD`. Читателю, знакомому с $\mathcal{AMS}$-TeX'ом, дальнейшее можно объяснить одной фразой: между `\begin{CD}` и `\end{CD}` надо поместить в точности тот же текст, что в $\mathcal{AMS}$-TeX'е пишут в аналогичном случае между `\CD` и `\endCD` (см. [5, гл. 19]). Для всех остальных удобнее пояснить правила набора коммутативных диаграмм на примере. Рассмотрим следующую диаграмму:

$$\begin{CD} 0 @>>> E' @>f>> E @>g>> E'' @>>> 0\\ @. @VVpV @VVqV @VVrV @.\\ 0 @>>> F' @>f>> F @>g>> F'' @>>> 0 \end{CD}$$

При подключенном пакете `amscd` она набирается следующим образом:

```
$$
  \begin{CD}
    0 @>>> E' @>f>> E @>g>> E'' @>>> 0\\
    @.   @VVpV    @VVqV   @VVrV      @.\\
    0 @>>> F' @>f>> F @>g>> F'' @>>> 0
  \end{CD}
$$
```

Первая строка в этой записи соответствует верхней строке диаграммы. Стрелка, направленная слева направо, задается конструкцией `@>>>` (а стрелка справа налево — конструкцией `@<<<`); если над стрелкой надо поставить какую-то надпись (например, просто букву), то нужно ее разместить между первым и вторым знаками неравенства; чтобы надпись получилась под стрелкой, надо ее разместить между вторым и третьим знаками неравенства.

Вторая строка задает вертикальные стрелки. Конструкция `@VVV` задает стрелку, направленную вниз; если справа от стрелки нужна надпись, то ее надо разместить между второй и третьей буквами `V` (чтобы надпись оказалась слева от стрелки, она должна быть, естественно, между первой и второй буквами `V`). Вертикальная стрелка, направленная вверх, задается конструкцией `@AAA` (буква `A` — максимальное приближение к устремленной вверх стрелке); справа и слева от нее также можно сделать надпись (аналогичным образом).

Конструкция `@.` задает «пустую» стрелку (в нашем случае — между двумя нулями); она необходима, чтобы LaTeX не сбился со счета, выясняя, в какие колонки ставить вертикальные стрелки.

Опишем работу окружения `CD` более аккуратно. Каждую коммутативную диаграмму окружение `CD` рассматривает как таблицу, состоящую из перемежающихся «горизонтальных» и «вертикальных» строк. Каждая «горизонтальная» строка состоит из формул, перемежающихся горизонтальными стрелками. Во всех горизонтальных строках должно быть одинаковое количество формул. Если некоторые из мест, предназначенных для формул, должны остаться пустыми, то на этом месте надо оставить пробел или, если вам так приятнее, написать `{}`. Между каждой парой формул должна быть стрелка. Если какие-то из этих стрелок не нужны, на их месте надо поставить `@.` («пустую» стрелку).

Каждая «вертикальная» строка состоит из вертикальных стрелок. Их должно быть столько же, сколько формул в любой из горизонтальных строк. Если какие-то из вертикальных стрелок не нужны, на их месте надо поставить `@.` (пустую стрелку).

Если надпись при стрелке, направленной вниз (и задаваемой, стало быть, конструкцией `@VVV`), сама содержит букву `V`, то нужно ее (надпись) взять в фигурные скобки — иначе TeX не сможет понять, какая из букв `V` относится к надписи, а какая — к обозначению стрелки. Аналогичные меры надо принять, если надпись при стрелке, направленной вверх, содержит букву `A` (а также, естественно, если надпись при горизонтальной стрелке содержит знак `>` или `<`, хотя ввиду математического смысла таких надписей последнее менее вероятно).

Наряду со стрелками, в коммутативных диаграммах встречаются горизонтальные и вертикальные «растянутые знаки равенства»:

$$\begin{CD}
A @= B @>>> C \\
@V{V_1}VV @AfAA @| \\
D @<<g< E @<<< F
\end{CD}$$

```
$$
  \begin{CD}
  A @=      B    @>>>   C  \\
  @V{V_1}VV  @AfAA    @| \\
  D @<<g<  E    @<<<   F
  \end{CD}
$$
```

Как видно из этого примера, такие знаки задаются конструкциями `@=` (горизонтальный) и `@|` (вертикальный). Обратите также внимание, как мы защитили фигурными скобками символ `V` в надписи к левой вертикальной стрелке.

Конструкция `\pretend` ... `\haswidth` системы $\mathcal{AMS}$TEX (см. книгу [5]) в LaTeX'е не поддерживается.

Математики знают, что в коммутативных диаграммах могут встречаться не только горизонтальные и вертикальные стрелки: бывают и наклонные, и изогнутые, и пунктирные... Возможностей пакета `amscd` для печати таких стрелок недостаточно; если вам нужны такие более сложные диаграммы, стоит воспользоваться стилевым пакетом XY-pic (см. приложение Д).

В «чистом» (без подключения стилевых пакетов) LaTeX'е набор диаграмм не предусмотрен. На самый крайний случай, если нет ни `amscd`, ни XY-pic'а, можно сделать так:

```
$$
  \begin{array}{ccccccccc}
  0&\longrightarrow & E' &
  \stackrel{f}{\longrightarrow}& E &
  \stackrel{g}{\longrightarrow} &
  E'' & \longrightarrow & 0\\
  &&\downarrow\lefteqn{p}&&\downarrow
  \lefteqn{q}&&\downarrow\lefteqn{r}\\
  0&\longrightarrow & F' &
  \stackrel{f}{\longrightarrow}& F &
  \stackrel{g}{\longrightarrow} & F''
  & \longrightarrow & 0
  \end{array}
$$
```

В результате получится почти такая же диаграмма, как в нашем первом примере (правда, буквы при вертикальных стрелках будут крупнее букв при горизонтальных, поскольку команда `\stackrel` уменьшает буквы). Единственное, что тут нуждается в пояснении, — команды `\lefteqn`. Они нужны для того, чтобы вертикальные стрелки с надписями были правильно центрированы. Если эти `\lefteqn`'ы опустить (и писать `p` вместо `\lefteqn{p}` и т. п.), то вертикальные стрелки с подписями окажутся не по центру, а сдвинутыми влево.

Для интересующихся объясним, в чем тут дело. В процессе верстки текста TEX учитывает, сколько места занимает тот или иной фрагмент формулы. В TEX'е предусмотрены специальные команды, позволяющие фальсифицировать эти данные. В частности, команда `\lefteqn` печатает формулу, являющуюся ее аргументом, но при этом сообщает TEX'у, что по горизонтали эта формула не занимает места вообще. Стало быть, с точки зрения TEX'а ширина элемента, стоящего во второй строке нашей таблицы, определяется только шириной стрелки, и при центрировании текст располагается так, чтобы именно стрелка была на равном расстоянии от краев, сколь бы длинна на самом деле ни была формула, стоящая в `\lefteqn`. Создатель TEX'а Дональд Кнут назвал такого рода приемы работы с TEX'ом «грязными трюками» (dirty tricks). Впрочем, при написании TEX'овских макропакетов используются трюки и похлеще.

4.4. Чего мы еще не сказали

Для читателя-математика того, что мы уже рассказали о наборе формул, должно быть, в принципе, достаточно, за одним важным исключением: при написании математической статьи полезно знать, как грамотно оформлять тексты теорем, определений и тому подобные вещи. Об этом у нас рассказывается в разд. 5.2 и 5.3 главы VII; вы можете прочитать это уже сейчас, пропуская непонятные места и справляясь при необходимости с разд. IV.5, посвященным рубрикации документа.

5. Тонкая настройка

В этом разделе мы рассмотрим некоторые более изысканные вопросы, связанные с набором математических формул. Рекомендуем читателям, не являющимся полиграфистами, пропустить в этом разделе мелкий шрифт (хотя бы при первом чтении).

5.1. Пробелы вручную

Бывают случаи, когда промежутки между символами в формулах, выбранные TeX'ом автоматически, выглядят неудачно. В этом случае в формулу можно включить команды, задающие промежутки в явном виде. Вот основные из них:

`\quad`	Пробел в `1em`: $|\quad|$
`\qquad`	Пробел в `2em`: $|\qquad|$
`\,`	«Тонкий пробел», или тонкая шпация: $|\,|$
`\:`	«Средний пробел»: $|\:|$
`\;`	«Толстый пробел»: $|\;|$
`\!`	«Отрицательный тонкий пробел»

Команда `\!` из этой таблицы уменьшает промежуток на столько же, на сколько команда `\,` его увеличивает.

В следующем примере собраны типичные случаи, когда в этих командах возникает нужда.

Пробелы надо корректировать в таких формулах, как $\int f(x)\,dx$, $\int\!\!\int f\,dx\,dy$ или $\sqrt{3}\,x$.

```
Пробелы надо
корректировать в
таких формулах,
как~$\int f(x)\,dx$,
$\int\!\!\int f\,dx\,dy$
или~$\sqrt{3}\,x$.
```

Команда `\qquad` полезна для отделения текста, входящего в формулу, от собственно формулы (см. с. 65). Для этих же целей можно использовать команду `\quad`, делающую пробел размером `1em`. Вместо `\int\!\!\int` лучше, конечно, подключить пакет `amsmath` и сказать `\iint`.

5.2. Размер символов в формулах

В большинстве случаев вам не приходится задумываться о том, какой размер будут иметь символы в формуле: TEX автоматически выбирает более мелкий шрифт для степеней, индексов, числителей и знаменателей дробей, созданных командой `\frac`, и т. п. Бывают, однако, случаи, когда в этот процесс автоматического выбора размера приходится вмешаться. Сейчас мы вкратце опишем, как TEX выбирает размеры символов в формулах и как можно на него при этом влиять.

При наборе формулы TEX в каждый момент руководствуется одним из следующих «стилей»:

displaystyle	«выключной» стиль
textstyle	«текстовый» стиль
scriptstyle	стиль для индексов
scriptscriptstyle	стиль для индексов к индексам.

«Выключной» и «текстовый» стили используют одинаковые шрифты, но формулы в текстовом стиле выглядят чуть скромнее (например, в выключном стиле верхние индексы поднимаются повыше, а нижние опускаются пониже, чем в текстовом; в текстовом стиле «пределы» операций записываются не сверху, а сбоку — ср. разд. 1.2). В стиле для индексов используются более мелкие шрифты, чем в выключном или текстовом (а в стиле для индексов к индексам — еще более мелкие). Выбираются стили набора формул следующим образом: выключная формула начинает набираться в выключном стиле, внутритекстовая — в текстовом стиле; далее, если в момент действия какого-то из стилей встретится команда `\frac` (или `\atop`), то для набора числителя и знаменателя TEX переключается на следующий по порядку стиль из вышеприведенной таблицы; если в момент действия выключного или текстового стиля встретится верхний или нижний индекс (показатель степени мы также рассматриваем как верхний индекс — в математическое содержание формул TEX не вникает), то этот индекс начинает набираться стилем для индексов; если индекс встретится в момент действия стиля для индексов или индексов к индексам, то набираться он будет в стиле «индексы к индексам». Например, при наборе формулы

$$x+\frac{y^2+3-z^{x^7-y^7}}{1+\cos^2 x}$$

TEX использует выключной стиль при наборе $x+$, текстовый стиль при наборе $y^2+3-z^{x^7-y^7}$ и $1+\cos^2 x$, стиль для индексов при наборе x^7-y^7 и двойки в показателе степени, и стиль для индексов к индексам при наборе семерок в показателях степени. Стиля `scriptscriptscriptstyle` и дальнейших не предусмотрено, так что индексы третьего и более высоких порядков набираются теми же шрифтами, что и индексы второго порядка (расстраиваться по этому поводу не надо, поскольку эти шрифты и так мелкие).

Если вы хотите изменить стиль набора формулы, можно в явном виде указать его с помощью TEX'овской команды, имя которой совпадает с английским названием этого стиля (`\displaystyle`...`\scriptscriptstyle`). Вот типичный пример, когда это может понадобиться. Предположим, в вашем тексте встречаются «цепные дроби». На с. 79 показано, какой неудачный результат выйдет, если набирать цепную дробь наивным образом. Теперь мы видим, что этот результат формально полностью согласуется с правилами TEX'а: вся формула, коль скоро она выключная, набирается в выключном стиле, стало быть числитель и первый из знаменателей будут уже в текстовом стиле, следующий знаменатель — как индексы, следующий — как индексы к индексам, и т. д. Если нет возможности воспользоваться командой `\cfrac` из пакета `amsmath`, то надо набирать так:

```
$$
  \frac{7}{25}=
  \frac{1}{\displaystyle
     3+\frac{1}{\displaystyle
          1+\frac{1}{\displaystyle
               1+\frac{1}{3}}}}
$$
```

Каждая из трех команд `\displaystyle` необходима для того, чтобы каждая из последующих дробей набиралась в выключном стиле, невзирая на то, что она стоит в знаменателе.

5.3. Фантомы и прочее

На с. 89 мы столкнулись с командой `\lefteqn`, позволяющей напечатать фрагмент формулы и при этом сообщить TEX'у, что отдельного места (по горизонтали) на этот фрагмент отводить не надо. Иногда бывает полезно сделать обратное: включить в формулу символ, который сам не печатается, но место занимает. Вот пример такой ситуации.

Команда `\sqrt` автоматически выбирает размер знака радикала таким образом, чтобы он точно соответствовал высоте подкоренного выражения, и это очень хорошо. Иногда, однако, такой автоматический выбор приводит к не очень удачным результатам:

В формуле $\sqrt{a}+\sqrt{d}$ два знака радикала имеют разные размеры.

```
В формуле $\sqrt{a}+
\sqrt{d}$ два знака радикала
имеют разные размеры.
```

Дело тут, конечно, в том, что буквы a и d имеют разную высоту. Чтобы сделать знаки радикала одинаковыми, TEX надо обмануть: добавить в подкоренные выражения по символу, который чуть выше, чем a или d, чтобы подкоренные выражения оказались одной высоты. Этот символ, естественно, не должен печататься и не должен занимать места по горизонтали (лишние пробелы под корнем нам тоже ни к чему). Такой невидимый символ генерируется TEX'овской командой `\mathstrut`:

В формуле $\sqrt{\mathstrut a}+\sqrt{\mathstrut d}$ оба знака радикала имеют одинаковые размеры.

```
В формуле $\sqrt{\mathstrut a}
+\sqrt{\mathstrut d}$
оба знака радикала
имеют одинаковые размеры.
```

Точнее говоря, `\mathstrut` — это невидимый символ, равный по высоте скобке (и не имеющий ширины.

Невидимый символ, создаваемый командой `\mathstrut`, является частным случаем TEX'овской конструкции «фантома». Именно, если в формуле вы напишете

`\phantom{`*какая-то формула*`}`

то результат будет такой же, как если бы эта самая «какая-то формула» была сначала напечатана по всем правилам TEX'а, а затем аккуратно стерта с бумаги. Пример:

Все мы знаем, что знак радикала выглядит так: $\sqrt{}$.

```
Все мы знаем, что знак
радикала выглядит
так:~$\sqrt{\phantom{x}}$.
```

Кроме того, можно создать «вертикальный фантом» формулы (по вертикали будет оставлено столько же места, сколько занимала бы формула, по горизонтали вертикальный фантом места не занимает). Создается вертикальный фантом командой `\vphantom`. В частности, команда `\mathstrut` — это сокращение для `\vphantom{()}` . Возможны, наконец, и горизонтальные фантомы, занимающие по горизонтали столько же места, сколько заняла бы формула, и не занимающие места по вертикали. Создаются они командой `\hphantom`:

На пустое место $\hphantom{\sin^2\alpha}$ можно вписать формулу вручную.

```
На пустое место
$\hphantom{\sin^2\alpha}$
можно вписать формулу вручную.
```

Для полноты картины скажем об еще одной экзотической команде, называемой `\smash`. Подобно команде `\lefteqn`, она печатает символ, но при этом говорит TEX'у, что он не занимает места по вертикали. С помощью этой команды (а так же с помощью `\lefteqn`) можно накладывать в формулах один символ на другой. Вот пример совместной работы команд `\phantom` и `\lefteqn`:

$$\rlap{$\overbrace{}$}1+\underbrace{2+3+4}$$

```
$$
  \lefteqn{\overbrace{
           \phantom{1+2+3}}}
  1+\underbrace{2+3+4}
$$
```

Поясним, как устроен исходный текст, давший такое перекрытие скобок. Верхняя фигурная скобка, созданная командой `\overbrace`, ставится не над самой формулой $1+2+3$, а над ее фантомом. В результате команда `\overbrace` печатает фигурную скобку над пустым местом. Далее, вся эта конструкция стоит, в свою очередь, в аргументе команды `\lefteqn`, вследствие чего

TEX считает, что места по горизонтали она не занимает. Поэтому формула $1+\underbrace{2+3+4}$ начинается с того же места, что и фантом формулы $1+2+3$; в результате $1+2+3$ попадает аккурат под верхнюю скобку! Все это, конечно, — еще один пример «грязного трюка» (см. с. 89).

Если бы формула была не выключная, а внутритекстовая, то этот трюк прошел бы не столь гладко. Дело в том, что команда `\lefteqn` всегда набирает формулы в `\displaystyle`, поэтому размер фантома, над которым ставилась скобка, мог в принципе не совпасть с размером реально печатаемого фрагмента формулы. Чтобы уж совсем себя обезопасить, следовало бы в этом случае аргумент команды `\lefteqn` начать с `\textstyle`.

5.4. Снова об интервалах в формулах

Сейчас мы обсудим вкратце, какими правилами руководствуется TEX при расстановке интервалов в математических формулах. В стандартных ситуациях мы об этом не задумываемся, а полностью доверяем LATEX'у. То, о чем мы будем говорить, пригодится, если мы пользуемся в формулах сложными конструкциями (например, конструируем знак двойного интеграла из двух знаков интеграла и «отрицательных пробелов», как на с. 90) и при этом не хотим подбирать верные интервалы экспериментально.

При наборе формулы TEX рассматривает ее как состоящую из частей одного из следующих типов:

Обыкновенный символ	например, `\alpha`
Бинарная операция	см. с. 47
Бинарное отношение	см. с. 47
Математический оператор	см. с. 49, 52
Подформула	например, `{x^2}`
Знак препинания	`,` или `;` или `\colon` или `\ldotp`
Скобка	

Здесь подформула — это любой фрагмент формулы, заключенный в фигурные скобки. Команда `\colon` задает двоеточие, рассматриваемое как знак препинания (двоеточие, набранное непосредственно, рассматривается TEX'ом как знак бинарного отношения), а команда `\ldotp` — точку, рассматриваемую как знак препинания (точка, набранная непосредственно, рассматривается как обыкновенный символ). К бинарным отношениям (с точки зрения TEX'а) относятся также все стрелки (с. 48) и фрагменты формул, создаваемые командой `\stackrel`. При расстановке пробелов в формуле TEX руководствуется тем, к какому из перечисленных типов относятся ее составные части: символы бинарных операций окружаются «средними пробелами» (теми, что вручную задаются командой `\:`), а символы бинарных отношений — «толстыми» пробелами (вручную, как мы помним, толстый пробел задается командой `\;`); впрочем, в стилях для индексов и индексов к индексам (см. предыдущий раздел) эти пробелы опускаются; после знака препинания в большинстве случаев ставится «тонкий» пробел, и т. д.[2] Подформула (т. е. фрагмент формулы, за-

[2]Иногда полезно поставить пустые фигурные скобки {}, чтобы создать фиктив-

ключенный в фигурные скобки) рассматривается TEX'ом почти так же, как обычный символ:

Сравните $2+3$ и $2{+}3$: во втором случае знак плюс является подформулой, а не символом бинарной операции.

```
Сравните $2+3$ и $2{+}3$:
во втором случае знак
плюс является подформулой,
а не символом бинарной операции.
```

Кстати, с этим приемом (поставить фрагмент формулы в фигурные скобки, чтобы он рассматривался как обычный символ) мы уже сталкивались на с. 31, когда обсуждали, как задать в TEX'е десятичную дробь. Мы не будем вдаваться в точные правила расстановки пробелов (они перечислены в книге [2]). Для нас сейчас важнее то, что TEX можно заставить рассматривать любой фрагмент формулы как бинарную операцию, бинарное отношение или математическую операцию: для этого надо применить команды `\mathbin`, `\mathrel` или `\mathop` соответственно. Вот примеры того, как работают эти команды.

Иногда возникает нужда в символе $\hat{\otimes}$, рассматриваемом как символ бинарной операции. Естественно, этот символ можно сгенерировать, если написать `\hat{\otimes}`, но тогда вокруг этого символа будут неправильные пробелы:

Хотелось бы, чтобы в формуле $E\hat{\otimes}F$ были такие же пробелы, как и в формуле $E\otimes F$.

```
Хотелось бы, чтобы в формуле
$E\hat{\otimes}F$ были такие
же пробелы, как и в
формуле $E\otimes F$.
```

Чтобы TEX рассматривал $\hat{\otimes}$ не как обычный символ, а как символ бинарной операции, надо сделать так:

В формуле $E\mathbin{\hat{\otimes}}F$ пробелы такие же, как и в $E\otimes F$.

```
В формуле
$E\mathbin{\hat{\otimes}}F$
пробелы такие же, как
и в $E\otimes F$.
```

Если символ $\hat{\otimes}$ встречается в вашей рукописи часто, то вам вряд ли понравится всякий раз делать по 23 нажатия на клавиши для его набора. В этом случае очень удобно ввести для него собственное сокращенное обозначение (посмотрите начало гл. VII по поводу того, как это сделать).

Типичный пример использования команды `\mathop` — определение имени операции, записываемой прямым шрифтом (см. с. 50, где мы давали определение функции tg). Обозначения такого типа встречаются в математических текстах очень часто, и набора команд для них, предусмотренного LATEX'ом (с. 49), вполне может не хватить; в этом случае, чтобы получить на печати, скажем, $\mathrm{Ext}^1(E,F)$, надо написать

```
$\mathop{\mathrm{Ext}}\nolimits^1(E,F)$
```

ный аргумент бинарной операции и тем самым обеспечить желательные пробелы; мы делали это в примере на с. 83.

Здесь \mathop необходимо для того, чтобы между Ext^1 и (E, F) автоматически вставлялся маленький дополнительный пробел, делающий формулу более читаемой:

Сравните $\sin x$ и $\mathrm{sin}x$.

```
Сравните $\sin x$ и $\mathrm{sin}x$.
```

Что же касается \nolimits, то эта команда необходима для того, чтобы в выключных формулах (точнее, в «выключном стиле» — см. разд. 5.2) верхние и нижние индексы к «оператору» записывались именно как индексы, а не над и под ним, как «пределы» (см. с. 53). Именно таким образом работает определенная в пакете amsmath команда \DeclareMathOperator.

А вот пример, когда TEX'у надо объяснить, что некоторый сложный символ есть символ математического оператора. Пусть нам понадобилась формула

$$\mathop{{\sum}'}_{x\in\Gamma} f(x).$$

Проблема тут в том, чтобы поставить штрих у знака суммы. Впрямую это сделать не удается:

$$\sum'_{x\in\Gamma} f(x).$$

```
$$
  \sum'_{x\in\Gamma}f(x).
$$
```

В самом деле, из сказанного на с. 54 вытекает, что наша запись равносильна такой:

```
$$
  \sum^\prime_{x\in\Gamma}f(x).
$$
```

и в этой записи штрих рассматривается как предел суммирования. Не будем, однако, отчаиваться, а просто создадим новый оператор «сумма со штрихом»:

```
$$
  \mathop{{\sum}'}_{x\in\Gamma}f(x).
$$
```

Можете проверить, что на сей раз все получается как надо. В этой записи очень существенно, что \sum взято в фигурные скобки: благодаря этому символ, генерируемый командой \sum, рассматривается TEX'ом просто как подформула, поэтому и штрих после него стоит где положено, а не там, где бывают пределы суммирования. Вся подформула {\sum}' передается в качестве аргумента команде \mathop, благодаря чему наш новый символ «сумма со штрихом» рассматривается как математический оператор и пределы суммирования (в выключной формуле) ставятся у него, где положено.

Здесь опять разумно создать сокращенное обозначение, которое заменяло бы эту громоздкую запись.

5.5. Вертикальные отбивки вокруг выключных формул

За размер вертикальных отбивок, автоматически создаваемых TEX'ом вокруг выключных формул, отвечают следующие TEX'овские параметры:

`\abovedisplayskip`	`\abovedisplayshortskip`	(перед формулой)
`\belowdisplayskip`	`\belowdisplayshortskip`	(после формулы)

В каждой из этих пар второй параметр относится к случаю, когда и формула, и соседняя с ней строка текста коротки (в таком случае расстояние между формулой и текстом должно быть поменьше). Если вы хотите уменьшить эти отбивки, скажем, на 30%, напишите в преамбуле

```
\abovedisplayskip=.7\abovedisplayskip
```

(и аналогично для трех остальных параметров).

Можно не только пропорционально изменять отбивки вокруг выключных формул, как в примере выше, но и присваивать этим интервалам конкретные значения. При этом следует иметь в виду следующее обстоятельство. Интервалы, создаваемые между текстом и формулами, обладают определенной растяжимостью, что помогает TEX'у выравнивать страницы по высоте (говоря на TEX'ническом языке, эти интервалы являются «клеем»), и присваивая абсолютное значение таким интервалам, желательно также задавать их растяжимость и сжимаемость; см. с. 147 (а более подробно — разд. VIII.3.3) по поводу того, как это делается.

5.6. Горизонтальные отбивки вокруг формул

Некоторые авторы и издатели считают, что математический текст выглядит понятнее, когда каждая формула окружена дополнительным пробелами справа и слева от нее[3]. Для этих целей в TEX'е предусмотрен параметр `\mathsurround` (см. разд. I.2.9 по поводу TEX'овских параметров). Значение этого параметра — размер дополнительного пробела, вставляемого по обе стороны каждой внутритекстовой математической формулы (этот пробел не добавляется перед формулой, попавшей при печати в начало строки, и после формулы, попавшей в конец строки). При запуске LATEX'а значение этого параметра равно нулю, так что расстояния между формулами и окружающим текстом такие же, как между словами в тексте. Можно, однако, присвоить параметру `\mathsurround` ненулевое значение. Например, команда `\mathsurround=2pt` (будучи включена в преамбулу) окружает каждую формулу дополнительными пробелами по 2 пункта с обеих сторон.

Если нужно организовать дополнительные горизонтальные отбивки вокруг какой-то одной формулы, можно поместить команду, присваивающую значение параметру `\mathsurround`, непосредственно в саму формулу (между ограничивающими ее знаками доллара, в любое место). Важно только не забыть сделать пробел после обозначения для единицы длины (скажем, `pt`).

[3]Это актуально для текстов на языках с латинской графикой. В русских текстах формулы обычно достаточно выделяются уже за счет того, что в них используется латиница, а в тексте — кириллица.

Глава III

Набор текста

1. Специальные типографские знаки

Большинство знаков препинания (точка, запятая, двоеточие и т. п.) набираются очевидным образом: точке в исходном тексте, например, соответствует типографская точка на печати. В этом разделе речь пойдет о знаках, требующих специального набора.

1.1. Дефисы, минусы и тире

При печати на пишущей машинке эти знаки по внешнему виду не различаются. В издательских системах, основанных на TEX'е, различают дефис - (по-английски hyphen), короткое тире – (en-dash), длинное тире — (em-dash) и знак минуса $-$ (обратите внимание, что он отличается от обоих тире).

Чтобы получить на печати дефис, короткое тире или длинное тире, надо в исходном тексте набрать один, два или три знака `-` соответственно. В русских текстах часто используют длинное тире в качестве тире как такового, а короткое тире — в сочетаниях типа «я вернусь через 2–3 часа» (в исходном тексте это выглядит как `через 2--3 часа`; обратите внимание на отсутствие пробелов вокруг тире). Длинное тире в русском тексте обычно окружают (следуя традиции) пробелами; в английском обычно пробелов не делают. См. также разд. 1.5 ниже по поводу альтернативы длинному тире.

Знак минуса, в отличие от короткого тире, встречается только в математических формулах, и там он, как вы помните, изображается просто знаком `-` (см. разд. I.3).

Любознательный читатель может спросить, как получается, что запись `жж` в исходном тексте дает на печати всего-навсего две буквы «ж», а запись `--` дает тире, которое шире, чем два дефиса. Ответ: TEX'овские шрифты так устроены, что некоторые последовательности подряд идущих символов заменяются на печати на новый знак (говоря более техническим языком, в этих шрифтах используются лигатуры).

Другой пример лигатур — это то, как выглядит в основных шрифтах сочетание букв fi: не так, как поставленные рядом f и i (fi).

Близкое к этому явление — так называемый кернинг, когда некоторые пары букв, стоящие рядом, на печати автоматически сближаются: сравните XO (полученное на печати естественным образом) и XO (набранное со специальной командой, убирающей кернинг).

1.2. Кавычки

В отличие от пишущей машинки, книжный набор использует различные знаки для открывающей и закрывающей кавычек (вместо нейтрального знака `"`). В английских текстах открывающая кавычка изображается во входном тексте двумя подряд идущими обратными апострофами, закрывающая — двумя апострофами.

The “definitions” are translations rather than explanations.

```
The ``definitions'' are translations
rather than explanations.
```

Образование знака кавычек из двух апострофов — еще один пример лигатуры.

В русских текстах употребляются кавычки типа «елочки» и „лапки“. В исходном комплекте TeX'овских шрифтов эти символы отсутствовали; при установке системы LaTeX вы должны выяснить, как именно они задаются в полученной вами русификации.

Если в тексте встречаются кавычки внутри кавычек, то, согласно типографским правилам, внутренние кавычки должны отличаться от внешних: в английских текстах снаружи ставятся двойные кавычки, задаваемые как `` ` ` `` и `''`, а внутри одинарные, задаваемые как `` ` `` и `'`; в русских текстах можно, например, снаружи поставить «елочки», а внутри „лапки“. Если при этом наружная и внутренняя кавычка соседствуют, их надлежит разделить дополнительным небольшим пробелом. В LaTeX'е этой цели служит команда `\,`. Пример (в русификации LaTeX'а, использованной при наборе этой книги, «елочки» задаются командами (с именами из русских букв!) `\лк` и `\пк`, а „лапки“ — командами `\glqq` и `\grqq`):

« „Карова“ или „корова“, как правильно?» — спросил он.

```
\лк\,\glqq Карова\grqq{} или
\glqq корова\grqq,
как правильно?\пк~---
спросил он.
```

Мы поставили символ ~ после закрывающих кавычек, чтобы тире заведомо напечаталось на той же строчке, что и предшествующее слово (см. с. 103 ниже).

1.3. Многоточие

На пишущей машинке многоточие — это три точки подряд (каждая из которых имеет стандартную ширину буквы). При наборе это не так: для многоточия есть специальная команда `\ldots` или `\dots`.

Вместо “...” пишем: Нет, что-то здесь не так…	`Вместо ''...'' пишем:` `Нет, что-то здесь не так\ldots`

(Отметим кстати, что команда `\dots` может встречаться и в формулах, где она — в зависимости от контекста — может давать многоточие в центре строки, как `\cdots`, или в низу строки, как `\ldots`.)

При подключении стилевого пакета `amsmath` (а это приходится делать очень часто) могут появиться нежелательные пробелы около многоточий, задаваемых команды `\dots` и `\ldots`. Случится это или нет, зависит от версии пакета `amsmath`, но самый простой способ справиться с этой проблемой, если это произошло — употреблять в тексте (но не в формулах!) вместо `\dots` или `\ldots` команду `\textellipsis`, никаких лишних пробелов не создающую.

1.4. Параграф, копирайт и прочее

Знак параграфа набирается с помощью команды `\S`, знак © — с помощью команды `\copyright`; о том, что знаки $ и & набираются с помощью команд `\$` и `\&`, мы уже говорили (см. разд. I.2.2); знак фунта стерлингов £ набирается с помощью команды `\pounds` или ее синонима `\textsterling`. Чтобы получить на печати знак №, можно подключить стилевой пакет `textcomp` (см. разд. 1.5 ниже), после чего этот знак можно будет набирать с помощью команды `\textnumero`. (Заодно подключение этого пакета может несколько улучшить вид знака параграфа.) Кроме того, в тексте можно использовать и любой из великого множества математических символов, если оформить его как математическую формулу:

Я $\heartsuit$ TeX.	`Я $\heartsuit$ \TeX.`

1.5. Экзотика

В поставку LaTeX’а входит стилевой пакет `textcomp`, открывающий доступ к большому числу типографских значков. Если его подключить (для чего надо сказать в преамбуле `\usepackage{textcomp}`), то становятся доступными следующие символы:

\textquotestraightbase	‚
\texttwelveudash	—
\textleftarrow	←
\textblank	␢
\textasteriskcentered	∗
\textfractionsolidus	⁄
\textoneoldstyle	1
\textthreeoldstyle	3
\textfiveoldstyle	5
\textsevenoldstyle	7
\textnineoldstyle	9
\textminus	−
\textmho	℧
\textohm	Ω
\textrbrackdbl	⟧
\textdownarrow	↓
\textborn	⋆
\textdivorced	⚮
\textleaf	🍃
\texttildelow	˷
\textasciibreve	˘
\textgravedbl	̏
\textdagger	†
\textperthousand	‰
\textbardbl	‖
\textcelsius	℃
\textdollar	$
\textcent	¢
\textflorin	ƒ
\textnaira	₦
\textpeso	₱
\textdong	₫
\textsterling	£
\textrecipe	℞
\textinterrobangdown	⸘
\textpilcrow	¶
\textdiscount	⁒
\textopenbullet	◦
\textlquill	⁅
\textbrokenbar	¦
\textasciidieresis	¨
\textordfeminine	ª
\textquotestraightdblbase	„
\textthreequartersemdash	—
\textrightarrow	→
\textquotesingle	'
\textdblhyphen	⹀
\textzerooldstyle	0
\texttwooldstyle	2
\textfouroldstyle	4
\textsixoldstyle	6
\texteightoldstyle	8
\textlangle	〈
\textrangle	〉
\textbigcircle	◯
\textlbrackdbl	⟦
\textuparrow	↑
\textasciigrave	`
\textmarried	⚭
\textdied	†
\textmusicalnote	♪
\textdblhyphenchar	⹀
\textasciicaron	ˇ
\textacutedbl	˝
\textdaggerdbl	‡
\textpertenthousand	‱
\textbullet	•
\textcolonmonetary	₡
\textdollaroldstyle	$
\textcentoldstyle	¢
\textwon	₩
\textguarani	₲
\textlira	₤
\textbaht	฿
\textyen	¥
\textinterrobang	‽
\texttrademark	™
\textnumero	№
\textestimated	℮
\textservicemark	℠
\textrquill	⁆
\textsection	§
\textcopyright	©
\textcopyleft	🄯

`\textlnot`	¬	`\textcircledP`	Ⓟ
`\textregistered`	®	`\textdegree`	°
`\textpm`	±	`\texttwosuperior`	²
`\textthreesuperior`	³	`\textasciiacute`	´
`\textmu`	µ	`\textparagraph`	¶
`\textperiodcentered`	·	`\textreferencemark`	※
`\textonesuperior`	¹	`\textonehalf`	½
`\textthreequarters`	¾	`\texttimes`	×
`\textdiv`	÷	`\textsurd`	√
`\textordmasculine`	º	`\textonequarter`	¼
`\texteuro`	€	`\textcurrency`	¤
`\textasciimacron`	¯		

Некоторые символы из этой таблицы нам уже знакомы, и их можно напечатать и без помощи пакета `textcomp` (например, можно сказать `$*$` вместо `\textasteriskcentered`; если набрать просто `*`, то звездочка напечатается в верхней части строки, а не в ее середине), но многие другие являются для нас новыми (кстати, команды, включающие в свое имя `florin`, `won` и т. п., суть обозначения для различных валют). Хотя в этой таблице и есть символ для «евро» (`\texteuro`), но его изображение довольно далеко от принятого (€); соответствующие шрифты и способы их подключения к LaTeX'у есть, но они пока не стандартизированы.

Полезно обратить внимание на команду `\textthreequartersemdash`: тире, задаваемое ей, может оказаться более приемлемым в русских текстах, чем «длинное тире», задаваемое как `---`.

Если какая-то из команд, перечисленных в этой таблице, встречается в вашем тексте часто, разумно определить для нее более короткое обозначение; в начале гл. VII объясняется, как это сделать.

1.6. Вызов символа по коду

Можно добраться до любого символа в текущем шрифте, если знать код этого символа. Для этих целей предназначена команда `\symbol`. Ее единственный обязательный аргумент — код символа. Для латинских букв и цифр эти коды совпадают с обычными ASCII-кодами:

Кошка по-английски будет cat.

```
Кошка по-английски будет
\symbol{99}\symbol{97}\symbol{116}.
```

Код символа можно указывать не только в десятичной системе, как в приведенном примере, но и в восьмеричной (тогда перед кодом надо поставить символ `'`) или шестнадцатеричной (перед кодом ставится символ ", «цифры» от A до F должны быть прописными буквами). Например, записи \symbol{122} , \symbol{'172} и \symbol{"7A} на печати дадут одно и то же: букву «z».

Если вы не знаете ASCII-кода нужного вам символа, можно в аргументе команды \symbol вместо номера поставить символы `\ и требуемый вам символ:

```
\sqrt                    {\ttfamily\symbol{`\\}\symbol{`\s}qrt}
```

Мы использовали в этом примере команду \ttfamily, включающую специальное начертание шрифта (см. с. 115), чтобы backslash выглядел на печати как \ (в большинстве TEX'овских шрифтов символу с этим кодом на печати соответствует нечто другое, см. таблицу в приложении B).

2. Подчеркивания, рамки

Чтобы подчеркнуть текст, используется команда \underline. У нее один обязательный аргумент — подчеркиваемый текст:

Это слово будет $\underline{\text{подчеркнуто}}$.

```
Это слово будет
\underline{подчеркнуто}.
```

Подчеркнутый текст должен умещаться в одной строке.

Чтобы взять часть текста в рамку, используется команда \fbox:

Два слова будут $\boxed{\text{в рамке.}}$

```
Два слова будут \fbox{в рамке.}
```

Команда \fbox позволяет взять в рамку только фрагмент текста, умещающийся в одну строку. Чтобы взять в рамку фрагмент, состоящий из нескольких строк, надо воспользоваться командами, о которых пойдет речь в гл. VIII.

При использовании подчеркивания и рамок могут пригодиться невидимые линейки, описанные в разд. III.10.3.

3. Промежутки между словами

3.1. Неразрывный пробел

Иногда необходимо обеспечить, чтобы два соседних слова не попали на разные строки. В этом случае между ними надо вставить «символ неразрывного пробела» ~. Такая необходимость возникает, например, в сочетаниях типа «на с. 5»: нельзя отрывать номер страницы от сокращения «с.». Вот еще примеры:

Число овец в стаде обозначено буквой x. Да здравствует император Франц-Иосиф I! Муж и жена — одна сатана. С кем вы, мастера культуры?	`Число овец в стаде обозначено буквой~$x$. Да здравствует император Франц-Иосиф~I! Муж и жена~--- одна сатана. С~кем вы, мастера культуры?`

В предпоследнем из этих примеров мы поставили неразрывный пробел, поскольку согласно отечественным полиграфическим правилам строка не должна начинаться с тире, а в последнем — потому что однобуквенное слово, начинающее предложение, не должно стоять последним в строке. Полный список русских однобуквенных слов (который может быть полезен, если вы автоматически расставляете символы `~` после них с помощью текстового редактора) содержится в магическом слове «АВИКОСУЯ».

3.2. Промежутки между предложениями

В обычном режиме TEX выравнивает справа строки абзаца, при необходимости делая переносы и слегка растягивая или сжимая промежутки между словами. Промежутки между предложениями при этом сами по себе шире и являются более растяжимыми, чем между словами внутри предложения. Посмотрите внимательно на следующий пример (из «Винни-Пуха»; для наглядности все промежутки равномерно растянуты):

North Pole. Discovered by Pooh. Pooh found it.

Такая печать соответствует английским типографским правилам, но в русских текстах промежутки между словами и между предложениями отличаться не должны. Чтобы так и было, следует включить в преамбулу команду `\frenchspacing`.

Если среди русского текста встречается фрагмент, написанный по-английски, то можно командой `\nonfrenchspacing` восстановить действие английского правила относительно межсловных промежутков. Когда английский текст кончится, надо восстановить действие российского правила командой `\frenchspacing` (другой вариант: заключить английский фрагмент вместе с командой `\nonfrenchspacing` в группу — по выходе из группы действие команды `\nonfrenchspacing` забудется).

Для читателей, которым необходимо набирать английские тексты, объясним более подробно правила расстановки промежутков в тех случаях, когда команда `\frenchspacing` не дается.

Чтобы отличить промежутки между словами от промежутков между предложениями, TEX применяет следующие правила:

1) Пробел увеличивается после:

- точки, вопросительного знака, восклицательного знака (в максимальной степени);
- двоеточия (несколько меньше);
- точки с запятой (еще меньше);
- запятой (совсем чуть-чуть).

2) Если последняя из букв, встретившихся перед одним из упомянутых в пункте 1 знаков препинания, была прописной, то пробел после этого знака препинания не увеличивается.

3) Если после одного из упомянутых в пункте 1 знаков препинания следует закрывающая скобка (круглая или квадратная) или закрывающие кавычки, а затем — пробел, то этот пробел увеличивается.

Смысл правила 2 в том, что точка после прописной буквы чаще всего обозначает не конец предложения, а конец чьих-то инициалов.

Как это и бывает обычно с «машинными эвристиками», сформулированные правила иногда приводят к неверным результатам: точка после строчной буквы может встретиться и в середине предложения, например, в сокращении, а точка после прописной буквы может, напротив, попасть в конец предложения. В этих случаях надо следующим образом помочь TEX'у сделать правильные пробелы:

- Если точка после строчной буквы не заканчивает предложения, то после нее следует поставить команду `\ ` (backslash с пробелом), генерирующую обычный пробел между словами (см. с. 20).
- Если точка (или любой другой из перечисленных в пункте 1 знаков препинания) после прописной буквы заканчивает предложение, то перед ней следует поставить команду `\@` — тогда пробел будет обычным образом увеличен.

Вот примеры:

If n is even (resp. odd), then $(-1)^n$ equals one (resp. minus one).
This research was supported by the NSF. The author is grateful to Prof. Smith.

```
If $n$ is even (resp.\ odd),
then $(-1)^n$ equals one
(resp.\ minus one).

This research was supported by
the NSF\@. The author is
grateful to Prof.~Smith.
```

Наконец, последнее правило относительно увеличения пробелов: если пробел задан как неразрывный с помощью символа `~`, то он не увеличивается, невзирая ни на какие предшествующие знаки препинания.

3.3. Установка промежутков вручную

Как команда «backslash с пробелом» `\ `, так и символ неразрывного пробела `~` генерируют пробел, но делать пробелы вручную с помощью набора чего-нибудь вроде `~~~` или `\ \ \ ` неразумно, поскольку эти пробелы, как правило, могут растягиваться или сжиматься ради выравнивания строк, и вы не сможете проконтролировать реальный размер пустого пространства, полученного таким способом.

Чаще всего требуется получить промежуток величиной в один или два `em` (см. с. 26). Для этого служат команды `\quad`, дающая промежуток в `1em`, и `\qquad`, дающая промежуток в `2em`. Команда `\enskip` дает промежуток, в два раза меньший, чем `\quad` (в стандартных шрифтах он равен ширине цифры). Про команду `\,` («backslash с запятой») уже шла речь в разд. 1.2 на с. 99.

Если необходимо задать промежуток с указанием конкретной длины, можно воспользоваться командой

`\hspace{`*длина*`}`

Если этот промежуток должен сохраняться также и в начале (или конце) строки, используется команда `\hspace*` вместо `\hspace`. Указание длины состоит из числа и названия единицы (см. с. 26), например, `\hspace{1.5cm}`

Пользуясь командами типа `\quad`, не сделайте лишнего пробела (разумеется, кроме того, который ограничивает имя команды, состоящей из букв). Вот примеры того, как надо и как не надо делать:

Здесь 1em промежутка.	`Здесь\quad 1em промежутка.\\`
Здесь 1em промежутка.	`Здесь\quad{}1em промежутка.\\`
Здесь 1em плюс еще немного.	`Здесь\quad{} 1em` `плюс еще немного.`

В этом примере используется еще команда `\\`, начинающая новую строку (см. подробнее на с. 122).

4. Диакритические знаки и прочее

Во многих языках используются буквы с дополнительными значками, размещающимися над или под буквой (они называются диакритическими знаками). Кроме того, в ряде языков, использующих латинский алфавит, есть специальные дополнительные буквы. В TEX'е имеются команды для набора таких букв из почти всех европейских языков. Команды для получения диакритических знаков собраны в нижеследующей таблице, где знаки проставлены, для примера, при букве «e».

Набрано	Вышло	Набрано	Вышло
`` \`e ``	è	`\'e`	é
`\^e`	ê	`\~e`	ẽ

\=e	ē	\.e	ė
\u{e}	ĕ	\v{e}	ě
\H{e}	e̋	\"e	ë
\c{e}	ȩ	\d{e}	ẹ
\b{e}	e̱	\r{e}	e̊
\t oo	o͡o		

В следующей таблице вы найдете команды для набора букв специального вида, а также перевернутых вопросительного и восклицательного знаков (последние используются в испанском языке).

Набрано	Вышло	Набрано	Вышло
\oe	œ	\OE	Œ
\ae	æ	\AE	Æ
\aa	å	\AA	Å
\o	ø	\O	Ø
\l	ł	\L	Ł
\i	ı	\j	ȷ
\ss	ß		
!‘	¡	?‘	¿

Обратите внимание на команды `\i` и `\j` в этой таблице: они нужны для того, чтобы ставить диакритические знаки над буквами i и j. Если просто сказать, допустим, `\=i`, то получится i̅, а это не то, что требуется. Правильно писать `\=\i`. Вот несколько примеров.

Chérie, ça ne me plaît pas!	Ch\'erie, \c{c}a ne me pla\^\i t pas!
Götterdämmerung	G\"otterd\"ammerung
¡Besa me mucho!	!‘Besa me mucho!

Для набора обычных русских текстов из всего этого великолепия нужны только две команды: `\'` (чтобы ставить ударения) и `\"` (для буквы ё, если в вашей русификации LaTeX’а для нее не отведено особого символа).

Команды `\c`, `\'` и т. д. имеют один обязательный аргумент — букву, над (или под) которой надо ставить диакритический знак. Читатель может заметить противоречие со сказанным на с. 23: ведь обязательный аргумент должен быть заключен в фигурные скобки, а в нашей таблице в записях вроде `` \`e `` или `\"e` фигурных скобок нет. На самом деле противоречия нет, просто на с. 23 было сказано не все: если у LaTeX’овской команды предусмотрен один обязательный аргумент и в исходном тексте после имени команды непосредственно следует буква, то в качестве аргумента будет воспринята именно эта буква (ср. с. 63). Так что можно, в принципе, не ставить букву в фигурные скобки и при командах наподобие `\c` или `\u` (но психологически приятнее, когда слово, которое на печати выйдет без пробелов, не будет содержать пробелов и в исходном тексте):

La façade est la façade.	`La fa\c{c}ade est la fa\c cade.`

Команды из второй таблицы аргументов не требуют; что же касается значков !` и ?`, то это вообще не команды, а лигатуры (см. текст мелким шрифтом на с. 99).

Над буквами в математических формулах также приходится ставить надстрочные знаки, но описанные в настоящем разделе команды для этого непригодны; команды, делающие это в формулах, описаны в разд. II.2.8.

В заключение нашего обсуждения диакритических знаков отметим вот что. Хотя описанные в этом разделе команды дают возможность набирать все знаки алфавита почти любого европейского языка, это еще не значит, что вы так просто сможете набирать длинные тексты на этих языках. Дело в том, что при отсутствии ориентированной на соответствующий язык таблицы переносов TeX будет, скорее всего, переносить слова в этих текстах неправильно (а слова с диакритическими знаками перенести вообще не сможет — см. разд. 6, с. 115). Таблица переносов для русского языка в любой русификации LaTeX'а имеется, а для английского языка такая таблица присутствует в TeX'е изначально. Если же вам придется набирать хотя бы абзац текста на каком-то другом языке, а соответствующей таблицы переносов у вас нет, то лучше переключиться на режим, в котором переносы вообще запрещены (как это сделать, будет рассказано в разд. 6 этой главы). Можно также попробовать воспользоваться стилевым пакетом `babel`, помогающим набирать тексты с правильными переносами на разных языках. См. приложение Е, разд. 6.

5. Смена шрифтов в тексте

В первой главе мы уже сталкивались с командами, изменяющими текущий шрифт. Сейчас мы рассмотрим их более подробно, но начнем с предупреждения, которое не требуется полиграфистам, но не повредит остальным читателям:

> Не увлекайтесь переключением шрифтов! Чем *меньше* различных видов шрифта использовано в тексте, тем легче его читать и тем красивее он выглядит.

5.1. Простые средства

Обычно шрифт, отличный от используемого в основной части текста, применяется для выделения каких-то частей этого текста. Например, шрифтом выделяют заголовки разделов; по этому поводу вам беспокоиться незачем, поскольку для таких выделений LaTeX выбирает шрифт автоматически (если, разумеется, вы оформляете разделы текста с помощью команд, описываемых в следующей главе, а не пытаетесь сделать это вручную).

Но может потребоваться выделить шрифтом не заголовок, который LaTeX делает сам, а какую-то выбранную вами часть текста — скажем, слово, на которое вы хотите обратить внимание читателя. Например, так выделено слово *меньше* в нашем предупреждении (и в этой фразе). Для этого использована команда `\emph`, аргументом которой служит выделяемое слово:

```
Например, так выделено слово \emph{меньше} в...
```

Выделяемое слово набирается курсивом, если текущий шрифт прямой, и прямым шрифтом, если текущий шрифт наклонный. Это полезно, так как в некоторых ситуациях (в заголовках, колонтитулах, в текстах «теорем» и т. п.), LaTeX выбирает шрифт за вас, и потому рекомендуется для выделения текста использовать в первую очередь именно эту команду.

Но можно указать шрифт и явно. Команда `\textit` набирает свой аргумент *курсивом* (так что в обычном тексте `\textit{слово}` неотличимо от `\emph{слово}`). Команда `\textsl` набирает свой аргумент *наклонным* шрифтом (обратите внимание на разницу между этим шрифтом и курсивом); команда `\textbf` — **полужирным** шрифтом. Есть еще команда `\texttt`, которая набирает свой аргумент шрифтом `типа пишущей машинки`. В этом шрифте все буквы имеют одинаковую ширину (как часто бывает на экране компьютера), и потому его часто используют для изображения программ, команд и сообщений операционной системы и т. п. В этой книге такой шрифт используется в примерах TeX'овских исходных текстов. Чтобы получить шрифт без засечек, надо воспользоваться командой `\textsf`. Шрифт «капитель», в котором строчные буквы представляют собой уменьшенные прописные, можно получить с помощью команды `\textsc`. Эти два шрифта обычно используются не для выделений в тексте, а для заголовков, подписей к рисункам и таблицам и др.

Вот, пожалуй, и все употребительные команды переключения шрифтов. Еще есть команда `\textup`, которая набирает свой аргумент прямым шрифтом (внутри наклонного или курсивного текста); часто ее применяют, чтобы набрать знаки препинания (скобки и др.) прямым шрифтом:

Таблица III.1. Смена размера.

Команда	Название размера
`\tiny`	Малюсенький z
`\scriptsize`	Очень маленький z (как индексы)
`\footnotesize`	Маленький (как сноски) z
`\small`	Мелкий z
`\normalsize`	Нормальный z
`\large`	Большой z
`\Large`	Очень большой z
`\LARGE`	Совсем большой z
`\huge`	Громадный z
`\Huge`	Грандиозный z

Рядом с $f(x)$ (*значением функции* f *в точке* x) *лучше использовать прямые скобки (а не курсивные).*

```
\textit{Рядом с $f(x)$
\textup{(}значением функции
$f$ в точке $x$\textup{)}
лучше использовать прямые
скобки (а не курсивные)}.
```

Теперь о размерах шрифтов. Команды изменения размера (полиграфисты бы сказали «кегля») указаны в табл. III.1. Такая команда (без аргументов) ставится в том месте, где нужно сменить размер шрифта, и действует до тех пор, пока размер не сменят вновь или не кончится группа.

Когда одно из слов набрано шрифтом другого кегля, это выглядит плохо.

```
Когда одно из {\scriptsize слов}
набрано шрифтом другого кегля, это
выглядит плохо.
```

Реальный размер шрифтов, задаваемых командами `\large`, `\small` и т. п., зависит от класса документа и классовых опций (разд. IV.1). В стандартных классах с основным шрифтом кегля 12 команды `\huge` и `\Huge` задают один и тот же размер (кегль 25).

Обратите внимание, что команды изменения размера текстового шрифта меняют и размер букв в формулах. Меняется и расстояние между строками по вертикали, если только не сделать одной распространенной ошибки. Если вы набрали шрифтом измененного размера (скажем, `\small` или `\footnotesize`) целый абзац, то в момент, когда TEX видит пустую строку (или команду `\par` — см. с. 142), этот шрифт не должен быть еще переключен на обычный, иначе интервалы между строками получатся неправильные. Вот пример того, как надо и как не надо делать:

Мы закрываем группу и возвращаемся к обычному шрифту только после пустой строки, завершающей абзац.
Вот шрифт обычного размера.

```
{\footnotesize Мы закрываем
группу и возвращаемся к обычному
шрифту только после пустой
строки, завершающей абзац.

}
Вот шрифт обычного размера.
```

Здесь мы вернулись к обычному шрифту раньше времени, и межстрочные интервалы оказались слишком велики.
Вот шрифт обычного размера.

```
{\footnotesize
Здесь мы вернулись к обычному
шрифту раньше времени, и
межстрочные интервалы
оказались слишком велики.}

Вот шрифт обычного размера.
```

Для любознательных объясним, откуда берется этот эффект. Расчет расстояний между строками происходит только после того, как абзац обработан. Поэтому, если TEX набирал абзац во время действия, скажем, команды `\small`, а команду «сверстать абзац» (например, пустую строку) увидел тогда, когда действие команды `\small` уже кончилось, то между строками, набранными мелким шрифтом, будут установлены те же расстояния, как между строками, набранными шрифтом обычного размера.

5.2. Подробности о шрифтах

Каждый из доступных в LATEX'е текстовых шрифтов характеризуется следующими четырьмя атрибутами: семейством (family), насыщенностью (series), начертанием (shape) и размером (size)[1]. Что такое размер, читатель разберется самостоятельно, а смысл остальных атрибутов следующий:

семейство означает примерно (но не в точности) то же, что отечественный термин «гарнитура»; в стандартной поставке определе-

[1]Это не вся правда. Более полную информацию о шрифтах можно найти в приложении B.

ны семейства rmfamily (шрифты с засечками), sffamily (шрифты без засечек) и ttfamily (шрифты типа «пишущая машинка»);

насыщенность определяет ширину и жирность шрифта. В стандартной поставке возможны насыщенности средняя (mdseries) и полужирная (bfseries);

начертание бывает прямое (upshape), курсивное (itshape), наклонное (slshape) и «капитель» (scshape).

Семейство, насыщенность, начертание и размер шрифта могут различным образом сочетаться. **Это предложение, например, набрано шрифтом семейства «без засечек» (sffamily), полужирной насыщенности (bfseries), прямого начертания (upshape) и размера large.**

Каждый из шрифтовых атрибутов можно менять независимо от остальных. Разберем, какие команды для этого предусмотрены.

Все команды для изменения размера вам уже известны: это десять команд, от `\tiny` до `\Huge`, перечисленных в предыдущем разделе. Каждая из команд для изменения остальных атрибутов существует в LaTeX'е в двух вариантах:

1) в виде команды без аргументов, меняющей атрибут текущего шрифта вплоть до того момента, пока он не будет изменен другой командой (или пока не закончится группа, если атрибут менялся внутри группы);

2) в виде команды с одним аргументом (помещаемым, как водится, в фигурные скобки), меняющей атрибут шрифта только у своего аргумента (т. е. у текста в фигурных скобках) — некоторые из этих команд (`\textit`, `\upshape` и т. д.) уже нам знакомы.

Имя команды без аргумента совпадает с английским названием соответствующего атрибута (например, `\sffamily` или `\scshape`); имя команды с аргументом состоит из слова `text`, к которому добавлены две буквы, описывающие атрибут (например, `\textsf` или `\textsc`).

В табл. III.2 перечислены все команды для смены атрибутов в обоих вариантах. Обратите внимание, что в трех строках этой таблицы шрифт совпадает с основным шрифтом текста: это строки, в которых стоят команды, устанавливающие семейство «с засечками», среднюю насыщенность или прямое начертание. Поскольку текущий шрифт и так обладает этими атрибутами, от соответствующей команды он не меняется.

Вот пример применения этих команд:

Таблица III.2. Смена начертания: команды с аргументами и без.

Без аргументов	С аргументом	На печати выйдет
`{\rmfamily Шрифт}`	`\textrm{Шрифт}`	Шрифт
`{\sffamily Шрифт}`	`\textsf{Шрифт}`	Шрифт
`{\ttfamily Шрифт}`	`\texttt{Шрифт}`	`Шрифт`
`{\mdseries Шрифт}`	`\textmd{Шрифт}`	Шрифт
`{\bfseries Шрифт}`	`\textbf{Шрифт}`	**Шрифт**
`{\upshape Шрифт}`	`\textup{Шрифт}`	Шрифт
`{\itshape Шрифт}`	`\textit{Шрифт}`	*Шрифт*
`{\slshape Шрифт}`	`\textsl{Шрифт}`	*Шрифт*
`{\scshape Шрифт}`	`\textsc{Шрифт}`	ШРИФТ

Сменим сначала *начертание, затем* `*семейство, затем*` `*размер, затем*` `насыщенность,` затем все вернем на место.

```
Сменим сначала \slshape
начертание, затем \ttfamily
семейство, затем \small размер,
затем \bfseries насыщенность,
затем \upshape\mdseries
\rmfamily\normalsize
все вернем на место.
```

Некоторым сочетаниям атрибутов никакого шрифта не соответствует. В этом случае затребованный, но отсутствующий шрифт заменяется на другой (по возможности, с близкими атрибутами). В нашем примере, в частности, не существует шрифта с атрибутами `\ttfamily` и `\bfseries`, поэтому LaTeX действует так, словно была дана команда `\mdseries`. О каждой такой замене выдается сообщение в процессе трансляции.

А вот пример, когда для смены атрибутов шрифта используются команды с аргументом:

Выберем **полужирный шрифт в *курсивном* начертании** (временно, конечно же).

```
Выберем \textbf{полужирный
шрифт в \textit{курсивном}
начертании} (временно,
конечно же).
```

Обратите внимание, что на фоне полужирного шрифта (`\bfseries`) команда `\textit` поменяла только атрибут «начертание», сменив его на курсивное.

После многочисленных изменений атрибутов шрифта хочется вернуться к обычному шрифту «одним махом», не устанавливая заново все четыре атрибута. Для этих целей предусмотрена команда `\normalfont`, переключающая шрифт на «нормальный» — основной шрифт документа. Наряду с ней есть, как водится, и команда с одним аргументом `\textnormal`, печатающая текст, являющийся ее аргументом, основным шрифтом.

В стандартных TEX'овских шрифтах (гарнитура Computer modern и ее русские аналоги) жирный шрифт, задаваемый командами `\bfseries`, `\textbf` и т. п., выглядит, по мнению многих, довольно неудачно. Можно сделать его несколько более приемлемым, если написать в преамбуле такую строку:

```
\renewcommand{\bfdefault}{b}
```

При этом **жирный шрифт** станет **менее широким**: обратите внимание на разницу в букве «ш» в словах «шрифт» и «широким». (Что такое `\renewcommand`, объясняется в главе VII.)

Если по какой-либо причине вы используете команды изменения атрибутов без аргумента, следует иметь в виду одну тонкость. При соседстве слова, набранного шрифтом с наклоном (курсивным в частности) и слова, набранного прямым шрифтом, последняя буква «наклонного» и первая буква «прямого» слова могут слишком сблизиться, что на печати выглядит некрасиво. Чтобы избежать этого явления, необходимо после последней буквы слова, которое будет набрано наклонным шрифтом, поставить команду `\/` ; она создаст после буквы небольшой дополнительный пробел (зависящий от шрифта и от буквы), который скомпенсирует наклон и предотвратит нежелательное сближение со следующей буквой:

You *and* I	`You {\itshape and} I\\`
You *and* I	`You {\itshape and\/} I`

(команда `\\` используется здесь для разрыва строки).

Если фрагмент текста, имеющий наклон, завершается точкой или запятой, то после них ставить `\/` не нужно: требуемый эффект достигается за счет места, занимаемого в тексте этим знаком.

Всё это, повторим, относится лишь к командам изменения атрибутов без аргумента; команды `\textit`, `\textsl` и `\emph` вносят нужную поправку автоматически.

Если команда `\/` поставлена между двумя символами, дающими на печати лигатуру (см. с. 99), то вместо лигатуры на печати получатся два этих символа по отдельности; если эту команду поставить в слове между двумя символами, между которыми в текущем шрифте предусмотрен кернинг, то кернинг между этими символами будет отключен.

5.3. «Старые» команды изменения шрифта

Наряду с описанными выше, в LaTeX'е пока сохраняются (для совместимости с LaTeX'ом 2.09) команды переключения шрифта, перечисленные

Таблица III.3. Смена начертания: старые команды.

Команда	Равносильна последовательности команд
`\bf`	`\normalfont\bfseries`
`\it`	`\normalfont\itshape`
`\sl`	`\normalfont\slshape`
`\sf`	`\normalfont\sffamily`
`\sc`	`\normalfont\scshape`
`\tt`	`\normalfont\ttfamily`
`\rm`	`\normalfont\rmfamily`

в таблице III.3. До некоторых шрифтов с помощью «старых» команд добраться невозможно (поскольку шрифты, получаемые с помощью этих команд, отличаются от «основного» не более чем в одном атрибуте).

6. Абзацы

Чтобы TeX сверстал абзац, никаких специальных усилий прилагать не нужно: достаточно оставить в исходном тексте пустую строку, указывающую TeX'у на конец абзаца. В этом разделе речь пойдет о тех «нештатных ситуациях», которые могут при этом возникнуть. Система TeX предоставляет множество способов реакции на эти ситуации; некоторые из них важны для всех пользователей, но бо́льшая часть — только для полиграфистов. Рекомендуем читателям, полиграфистами не являющимися, пропускать весь мелкий шрифт в этом разделе.

6.1. Overfull и underfull

Обычно абзацы делаются выровненными по правому краю; при необходимости промежутки между словами растягиваются или сжимаются, а в словах делаются переносы. При этом и для сжатия, и для растяжения промежутков между словами есть пределы, которые TeX старается не превышать.

Если это удается, то получается аккуратный абзац, но нам сейчас интереснее, что происходит в том случае, когда это не удается.

Прежде всего заметим, что «предел сжимаемости» строки не превышается TeX'ом ни при каких условиях[2]; что бы ни было, строки не станут более тесными, чем им позволяют параметры шрифта. Поэтому строки, которые не удалось ужать, остаются чересчур длинными (при

[2]Если не предпринимать для этого специальных усилий.

этом, естественно, они выходят на поля документа). С другой стороны, предел растяжимости, за неимением лучшего, может быть превышен. При этом получается то, что полиграфисты называют жидкими, или разреженными строками:

Весьма   и   весьма   разреженная   строка.

О каждой из этих двух неприятностей TeX сообщает в процессе трансляции (эти сообщения появляются на экране и, кроме того, записываются в `log`-файл — специальный файл с тем же именем, что у файла, который обрабатывался системой, и расширением `log`).

Предположим, вы получили строку, выходящую на поля (в нашем примере она отмечена черным прямоугольником):

> Еще одно правило относительно увеличения пробелов: если
> пробел задан как неразрывный, то он не увеличивается, невзирая ни на какие предшествующие знаки препинания.

При этом выдается на экран и записывается в файл с расширением `log` сообщение наподобие такого:

```
Overfull \hbox (3.2673pt too wide) in paragraph at lines 8--13
[]\OT1/cmr/m/n/10.95 Еще од-но пра-ви-ло от-но-си-тель-но
уве-ли-че-ния про-бе-лов: если
 []
```

Давайте разберем, что в нем написано. Сначала идет надпись «`Overfull \hbox`», указывающая, что произошло «переполнение» (overfull) строки. В скобках указано, на какое именно расстояние строка выходит за край: на 3.2673 пункта. (Напомним, что пункт примерно равен одной трети миллиметра — см. с. 26.) Далее сказано, что переполнение произошло при верстке абзаца (слова «`in paragraph`»), а затем указаны номера строк исходного файла, в которых был записан этот абзац.

Наконец, в этом сообщении приведен фрагмент неудачной строки вблизи ее конца (конца не в исходном тексте, а на печати)[3]. Обратите внимание, что в некоторые слова вставлены дефисы: они показывают те места, в которых TeX в принципе мог бы сделать переносы. Если вглядеться повнимательнее, то станет ясна и причина катастрофы: в слове `если`, которым заканчивается строка, дефиса не стоит вообще; значит, программа не смогла найти подходящего места для переноса и оказалась перед неприятным выбором: либо перенести это «если» целиком на другую строку (что, видимо, вызвало бы проблемы в других

[3]Загадочный набор символов, предшествующий фрагменту строки, характеризует текущий шрифт; см. приложение В.

местах), либо оставить его на этой строке и создать overfull. Выбрано было второе (ниже мы объясним, как можно в какой-то мере управлять этим выбором).

Сообщения о разреженных строках выглядят так:

```
Underfull \hbox (badness 1142) in paragraph at lines 885--892
[]\OT1/cmr/m/n/10.95 Некоторым со-че-та-ни-ям атри-бу-тов
ни-ка-ко-го ре-аль-но-го шриф-та
 []
```

Главных элементов в этих сообщениях три:

- само слово `Underfull`, указывающее, что речь идет о разреженной строке;
- указание на строки исходного текста, в которых находится абзац с разреженной строкой (в нашем случае 885–892);
- численная характеристика того, насколько разрежена строка (по-английски это число называется badness). В нашем случае это число равно 1142; вскоре мы обсудим, что оно значит.

Итак, мы выяснили, какие могут быть неприятности при верстке абзацев и как TeX о них сообщает. Вся оставшаяся часть этого раздела посвящена тому, как с этими неприятностями бороться.

6.2. Борьба с переносами

Вероятно, читатель был шокирован тем, что в нашем примере с overfull'ом TeX не смог найти, где сделать перенос в слове «если». Дело, в частности, в том, что обычно TeX не делает переносов, при которых от слова отрываются две последние буквы. Это соответствует нормам английской орфографии, но для русского языка такое ограничение неуместно. Поэтому при работе с русскими текстами можно (и нужно) установить такой режим, в котором перенос двух последних букв допустим. Устанавливается этот режим с помощью изменения параметра `\righthyphenmin` (см. с. 25 по поводу TeX'овских параметров). Значение этого параметра — целое число, равное наименьшему количеству букв в слове, которые можно переносить на следующую строку. Стало быть, если написать в файле

```
\righthyphenmin=2
```

то при обработке всех абзацев, кончающихся после этой команды, переносы с отрывом двух последних букв будут разрешены. Если вам встретится абзац английского текста, то перед завершающей его пустой строкой дайте команду

```
\righthyphenmin=3
```

дабы не сделать неверных переносов в английских словах.[4]

Общее облегчение режима переносов с помощью `\righthyphenmin` может, тем не менее, не помочь преодолеть встретившийся вам overfull: не исключено, что TeX действительно не знает, как перенести какое-то слово. Таких слов не очень много, но они встречаются. Кроме того, TeX не делает автоматических переносов в словах, содержащих диакритические знаки (например, если в слове указано ударение или если буква ё задана в тексте как `\"e`)[5], а также в словах, в которых присутствуют наряду с буквами цифры, знаки препинания (не в конце слова) и т. п. Далее, если в слове присутствует дефис, то TeX сможет сделать в нем перенос только на месте этого дефиса (слово «генерал-губернатор» будет перенесено так, что «генерал-» останется в конце строки, а «губернатор» начнет следующую строку). Что делать в таких случаях, когда автоматически вставляемых TeX'ом переносов не хватает?

Совет номер один — попробуйте немного отредактировать абзац. Обычно после небольших изменений в абзаце неприятности с переносами исчезают; качество текста при этом тоже зачастую улучшается (с точки зрения языка, а не TeX'а).

Пусть, однако, улучшать изложение дальше некуда, а абзац все равно получается неудачный. Что еще можно сделать, чтобы избавиться от переполнения?

Если TeX не может перенести слово, перенос которого по правилам русского языка возможен, то есть два способа указать TeX'у на это обстоятельство.

Во-первых, существует «одноразовый» способ указать TeX'у места переносов в слове. Это делается с помощью команды `\-` таким, например, образом:

```
тво\-р\'ог
```

Команда `\-` означает, что данное слово можно переносить в тех и только тех местах, где стоят знаки `\-` (хотя бы и вопреки тому, что диктует TeX'овский алгоритм переноса). Она годится для любых слов (с диакритическими знаками, цифрами и т. д.). Однако при этом TeX

[4]Сказанное в этом абзаце относится к вариантам русификации, где русские и английские слова включены в общую таблицу переносов (такова, например, русификация, использованная при подготовке этой книги и описанная в приложении Е). В других случаях могут потребоваться явные команды переключения языков (которые могут также заодно менять `\righthyphenmin`).

[5]Если вы пользуетесь пакетом `babel` вместе с так называемыми `EC`-шрифтами и соответствующими «форматными файлами», то слова с диакритическими знаками иногда могут переноситься.

не запоминает, какие слова и в каких местах позволила ему перенести команда `\-`. Если, например, то же слово «творóг» встретится в тексте еще раз, места переносов придется указывать заново.

Во-вторых, если слово-исключение встречается в тексте неоднократно, имеет смысл указать это TEX'у «раз и навсегда» (в пределах данного документа). Для этого предназначена команда `\hyphenation`. В качестве ее аргумента указываются слово или слова, в которых дефисами обозначены разрешенные места переносов. Например:

```
\hyphenation{вклю-чен об-ласть}
```

Теперь слова «включен» и «область» всегда будут переноситься так, как было указано (хотя бы и вопреки тому, что диктует алгоритм переноса). Если в слове, указанном в качестве аргумента команды `\hyphenation`, дефисов не поставить, то это будет означать, что переносить его вообще нельзя. Разумное место для команды `\hyphenation` — преамбула документа.

Слова, указанные в аргументе команды `\hyphenation`, должны быть разделены пробелами (конец строки — тоже пробел, так что слова можно располагать и в нескольких строках). Пустой строки в аргументе `\hyphenation` быть не должно. В качестве аргумента команды `\hyphenation` нельзя указывать слова с диакритическими знаками или небуквенными символами.

Слова в исходном тексте, в которые вставлены `\-`, будут переноситься именно там, где указано этими командами, невзирая на то, что говорит команда `\hyphenation`.

Иногда в тексте встречаются пары слов, объединенные знаком / . По общим TEX'овским правилам такое «слово» перенесено быть не может. Если вместо символа `/` использовать команду `\slash`, то станет возможен перенос, при котором на одной строке останется первое слово и символ /, а на второй строке — второе слово:

Перенаправление ввода/ вывода — одна из характерных черт систем типа UNIX/Linux.	`Перенаправление ввода\slash вывода~---` `одна из характерных черт систем типа` `UNIX\slash Linux.`

Бывают случаи, когда избавиться от переполнения не помогают даже идеально расставленные переносы: например, если в конец строки попадает слово, которое переносить нельзя (скажем, «гвоздь»), или не допускающая переноса математическая формула. Что делать в таких случаях, рассказано в последующих разделах.

6.3. Команда `\sloppy` и параметр `\emergencystretch`

Существует простой и грубый способ раз и навсегда избавиться от переполнений. Для этого достаточно включить в преамбулу файла команду `\sloppy` — больше сообщений о слишком длинных строках вы, скорее всего, не увидите. Для черновых распечаток команды `\sloppy`, помещенной в преамбулу, чаще всего бывает достаточно. При изготовлении оригинал-макета, однако же, доверяться ей полностью было бы рискованно, так как иногда в этом режиме могут появиться недопустимо разреженные строки. Разумнее задать эту команду не в преамбуле, а перед концом того абзаца, в котором произошел overfull (см. ниже по поводу того, как это сделать), и посмотреть, что из этого получится.

Чтобы отменить действие команды `\sloppy`, надо либо вернуться в обычный режим с помощью команды `\fussy`, либо давать команду `\sloppy` внутри группы, с тем, чтобы этот режим кончился по выходе из группы. В любом случае необходимо понимать, на какие участки текста распространяется действие команд типа `\sloppy`, влияющих на вид абзаца. Правило таково:

> *режим верстки абзаца определяется в тот момент, когда* TEX *читает пустую строку, завершающую абзац.*

В частности, если вы решили дать команду `\sloppy` внутри группы, то для того чтобы она подействовала на абзац, необходимо, чтобы закрывающая фигурная скобка шла *после* пустой строки, завершающей абзац. Вот пример того, как надо действовать в таких случаях:

Черепаховый суп — изысканное деликатесное и диетическое блюдо

```
Черепаховый суп~--- изысканное
деликатесное и диетическое блюдо
{\sloppy

}
```

А вот — типичная ошибка начинающего:

Черепаховый суп — изысканное деликатесное и диетическое блюдо

```
{\sloppy
Черепаховый суп~--- изысканное
деликатесное и диетическое блюдо}
```

К моменту, когда TEX увидел пустую строку, группа завершилась, поэтому TEX получил команду «разбить абзац на строки», находясь в стандартном режиме, и вместо желаемых разреженных, но не выходящих за край строк произошло переполнение (в первой строке).

Для более тонкого управления выбором между разреженными строками и overfull'ами используется параметр со значеним длины `\emergencystretch`.

Его точный смысл мы объясним ниже, а для начала скажем, что если установить его значение равным примерно 20–30 пунктам, т. е. написать, например,

```
\emergencystretch=25pt
```

то в случае, когда без переполнений сверстать абзац не удается, TEX попробует сделать *все* строки абзаца более разреженными (тем более разреженными, чем больше величина этого параметра). Точную величину `\emergencystretch` надо подбирать экспериментально.

6.4. Ручное управление разрывами строк

Иногда возникает необходимость повлиять на то, в каком месте TEX начинает новую строку. Для этой цели есть соответствующие команды, с одной из которых мы уже встречались — это «неразрывный пробел», запрещающий разрыв строки между двумя словами.

Иногда надо обеспечить, чтобы в каком-то слове не делалось переносов, причем не вообще никогда (тогда разумно применить команду `\hyphenation`), а только в данном месте. Можно добиться этого, например, с помощью команды `\mbox`, написав так:

Параметр **filename** задает имя файла.	`Параметр` `\mbox{\textbf{filename}}` `задает имя файла.`

Команда `\mbox` имеет один обязательный аргумент: в фигурных скобках может находиться любой текст, укладывающийся в одну строку (в том числе, как вы заметили, с командами переключения шрифта и т. п.); TEX будет рассматривать содержимое `\mbox`'а как одну большую букву и тем самым, конечно, не сможет разорвать его между строками.

Вы уже встречались с командой `\mbox`, если прочли в предыдущей главе раздел о включении текста в формулы; более подробно мы ее рассмотрим в главе о «блоках».

Теперь посмотрим, что делать, если вам понадобилось насильно разорвать строку в каком-то месте, не начиная при этом нового абзаца. Для этого есть несколько способов, в зависимости от того, что вы хотите получить. Один из вариантов — воспользоваться командой `\\` и получить возможно не доходящую до края, но не растянутую строку:

Эта строка была разорвана. Справа осталось пустое место, но зато строка не разреженная.	`Эта строка\\ была разорвана.` `Справа осталось пустое` `место, но зато строка` `не разреженная.`

Можно также воспользоваться командой `\linebreak`; при этом оборванная строка будет выровнена по правому краю, даже если ради этого ее придется растянуть:

Эта строка была
разорвана. Она выровнена по правому краю, но для этого ее пришлось безбожно растянуть.

```
Эта строка была\linebreak
разорвана. Она выровнена по
правому краю, но для
этого ее пришлось
безбожно растянуть.
```

Если строка действительно окажется разреженной, то вы получите сообщение об этом во время трансляции. Если абзац длинный, а команда `\linebreak` расположена не слишком близко к его началу, то скорее всего разреженных строк не будет.

Команда `\\` допускает и необязательный аргумент (см. с. 23): если в квадратных скобках указать какое-то расстояние (в TeX'овских единицах длины — с. 26), то после оборванной строки будет оставлено это расстояние (по вертикали). Пример:

Разорвем строку

и оставим место.

```
Разорвем строку\\[5pt] и
оставим место.
```

При использовании команды `\\` с необязательным аргументом бывает удобно вместо расстояния в явном виде указать один из следующих параметров:

`\smallskipamount`	маленький вертикальный пробел;
`\medskipamount`	вертикальный пробел побольше;
`\bigskipamount`	еще больше.

Точный размер этого пробела зависит от класса документа (и «классовых опций»); на с. 146 изображена величина соответствующих пробелов в стандартных классах со шрифтом кегля 11.[6]

Команда `\\` имеет и вариант со звездочкой (см. разд. I.2.8); если бы мы написали `\\*` или `\\*[`*расстояние*`]`, то эффект был бы тот же, что и без звездочки, и к тому же было бы запрещено заканчивать страницу на оборванной строке.

У команды `\linebreak` также может присутствовать необязательный аргумент. При этом команда `\linebreak[`n`]` указывает, что в данном месте желателен переход на новую строку, причем n указывает «силу» этого желания (n должно быть целым числом от 0 до 4). Если $n = 4$, то это полностью

[6]При печати книг оригинал-макет часто уменьшают; в этом случае, естественно, пропорционально изменяются и промежутки. Вот для сравнения промежуток, который в оригинал-макете имел размер в один сантиметр: | |

равносильно `\linebreak` без необязательного аргумента, если $n = 0$, то это означает только, что строку в данном месте разрешается разорвать (так что применять эту команду с аргументом 0 между словами обычно бессмысленно); когда n возрастает от 1 до 3, команда `\linebreak[`n`]` «усиливает давление» на TEX'овский алгоритм верстки абзаца, делая для него разрыв в указанном месте все более выгодным, невзирая на возможное появление разреженных строк.

Есть также команда `\nolinebreak`, действующая противоположно; она также может принимать необязательный аргумент — целое число от 0 до 4. Будучи заданной с аргументом 4, эта команда запрещает разрыв строки в указанном месте. Когда ее необязательный аргумент возрастает от 1 до 3, TEX начинает рассматривать разрыв строки в указанном месте как все менее желательный, даже невзирая на то, что из-за отказа от этого разрыва могут появиться разреженные строки. Команда `\nolinebreak[0]` равносильна, как это ни странно, команде `\linebreak[0]`. Команду `\nolinebreak` надо давать непосредственно после слова и до пробела, иначе она не сработает.

Для простых приложений, о которых идет речь в этой главе, команда `\nolinebreak`, как правило, не нужна: чтобы запретить разрыв, гораздо удобнее «неразрывный пробел». Команда `\nolinebreak` иногда бывает полезна при разработке собственных макроопределений, о чем сейчас говорить преждевременно.

6.5. Абзацы без выравнивания и переносов

Можно перевести TEX в режим, при котором он вообще не будет пытаться выравнивать текст по правому краю и не будет (почти никогда) делать переносов. Для этого служит команда `\raggedright`. Ее можно дать как в преамбуле, так и внутри документа; в любом случае, чтобы она подействовала на абзац, необходимо, чтобы ее действие не прекратилось до того, как TEX'ом будет прочтена пустая строка, завершающая абзац (ср. выше обсуждение команды `\sloppy`). Вот пример:

Этот абзац мы сверстали без
выравнивания и переносов.
Может быть, вид и не очень
аккуратный, зато без overfull'ов.

```
Этот абзац мы сверстали
без выравнивания и переносов.
Может быть, вид и не очень
аккуратный, зато без
overfull'ов.{\raggedright

}
```

Команда `\raggedright` в том виде, как она представлена в LATEX'е, делает абзацный отступ равным нулю, поскольку предназначена для оформления текста в виде так называемого «флагового набора». В приведенном выше примере этого не произошло, поскольку команда `\raggedright` была выполнена после начала абзаца, когда абзацный отступ уже был определен; если, однако, записать ее в преамбулу, то отступ будет равен нулю для всех абзацев. Если вам это не нравится, но выравнивать текст по правому краю все-таки не хочется, можно после `\raggedright` записать в преамбуле команду, устанавливающую значение абзацного отступа `\parindent` (см. с. 25; в стандартных классах значение этого параметра равно примерно `1.5em`).

6.6. Более тонкая настройка

Режимы, задаваемые командами `\sloppy` и `\fussy`, представляют собой две крайности. Здесь мы расскажем вам о более аккуратных способах управления разбиением на строки.

Параметр `\hfuzz`. Если вы получаете слишком много сообщений о переполнениях, можно попросить TeX вообще не считать слишком длинными те строки, которые выдаются за край не очень сильно. Для этих целей предусмотрен параметр `\hfuzz`. Например, команда

```
\hfuzz=2.5pt
```

указывает, что как overfull будут восприниматься лишь те строки, которые выступают за край более, чем на два с половиной пункта. В обычном режиме значение параметра `\hfuzz` равно одной десятой пункта.

Если `\hfuzz` равен примерно 0.5 пункта, то получается приемлемый для ординарных изданий результат. Дело в том, что на фоне идеально выровненных абзацев одна выдающаяся на 0.5 пункта строка смотрится хуже, чем длинный текст, где все абзацы выровнены не идеально, а «с точностью до 0.5 пункта».

Мера разреженности строки. Как вы помните, в сообщении TeX'а о разреженной строке фигурирует такая мера разреженности строки, как «badness». Посмотрите, как выглядят на печати разреженные строки с различными значениями этой меры:

Как может выглядеть разреженная строка.	badness = 0
Как может выглядеть разреженная строка.	badness = 142
Как может выглядеть разреженная строка.	badness = 740
Как может выглядеть разреженная строка.	badness = 1803
Как может выглядеть разреженная строка.	badness = 5133
Как может выглядеть разреженная строка.	badness = 10000

У последней из наших строк значение badness равно 10000. Если растянуть пробелы в строке еще сильнее, то badness уже не увеличится, а останется равной 10000: с точки зрения TeX'а, такие разреженные строки настолько плохи, что нет смысла делать различие между ними.

Для интересующихся объясним подробнее, как вычисляется badness. Как мы уже говорили в разд. 3.2, промежутки между словами в тексте не фиксированы, а могут растягиваться или сжиматься. Каковы эти пробелы и насколько они могут растягиваться, зависит от шрифта (для примера: у основного шрифта кегля 10 обычный промежуток между словами равен примерно 3.33 пункта, а его растяжимость составляет 1.67 пункта; промежуток же между предложениями в этом шрифте равен 4.44 пункта и имеет растяжимость в 5 пунктов)[7]. Когда TEX растягивает строку с целью выравнивания, он находит сумму «пределов растяжимости» всех промежутков — это «предел растяжимости» строки — и вычисляет, насколько требуемая длина строки больше «естественной» (определяемой размерами слов и нерастянутых промежутков между словами) — это «требуемое растяжение» строки. Отношение «требуемого растяжения» к «пределу растяжимости» строки определяет, насколько разреженной получится строка. Традиционно это отношение обозначается буквой r. Практически в качестве меры разреженности используется не само число r, а число $100r^3$ — это и есть badness. Если даже окажется, что $100r^3 > 10000$, badness все равно будет считаться равной 10000: строки, для которых отношение r больше или равно 4.7 (примерно при этом значении получается 10000), рассматриваются TEX'ом как одинаково плохие.

В том счастливом случае, когда требуемая длина строки совпадает с естественной, мера разреженности равна нулю; если мера разреженности не превосходит 100, то растяжение строки не превосходило предела; на самом деле даже строки, мера разреженности которых не превосходит 200, выглядят всё еще хорошо, хотя они уже и рассматриваются TEX'ом как слегка разреженные: TEX старается, чтобы такая «слегка разреженная» строка не попалась в абзаце рядом со строкой, в которой промежутки между словами сжимались. Сообщения об underfull'е появляются, когда badness превосходит 1000.

Теперь мы можем объяснить точный смысл параметра `\emergencystretch`. Если при верстке абзаца не удалось избежать переполнения, то — при условии, что значение `\emergencystretch` отлично от нуля, — TEX делает еще одну попытку, при которой в процессе перебора вариантов разбиения абзаца на строки (и вычислений соответствующих значений badness) к «пределу растяжимости» каждой из строк прибавляется значение `\emergencystretch`.

Параметр `\tolerance`. Теперь в нашем распоряжении есть все необходимые понятия, чтобы объяснить, как TEX выбирает между разреженной строкой и переполнением (см. с. 117)

При разбиении абзаца на строки TEX *никогда* не создает строки, мера разреженности (badness) которых больше, чем значение TEX'овского параметра, называемого `\tolerance`. При невозможности удовлетворить этому условию создаются строки, выходящие за край: возникает overfull. С другой стороны, если мера разреженности строки не превосходит значения `\tolerance`, то будет создана именно столь разреженная строка, но не overfull.

В отличие от некоторых других систем компьютерной верстки, TEX никогда не растягивает и не сжимает отдельное слово.

В стандартном режиме значение параметра `\tolerance` равно 200. Если установить значение `\tolerance` равным 10000, т. е. максимально возможному, то может получиться так, что одна из строк абзаца окажется совершенно ужасной: TEX вложит в нее «всю разреженность», чтобы не увеличивать то число, которое TEX минимизирует при переборе различных вариантов разбиения абзаца на строки (грубо говоря, это число тем больше, чем больше

[7]Кроме того, промежутки могут и сжиматься; пока речь идет только о жидких строках, это несущественно.

разреженных строк). Поэтому разумным решением во многих случаях будет увеличить значение `\tolerance`, но не до максимума, а до более разумной величины (скажем, 300 или 400). После этого TeX, с одной стороны, получит бо́льшую свободу действий, а с другой — не сможет создавать абзацы, в которых все строки, кроме одной, приемлемы, а одна разрежена до безобразия.

В частности, именно так работает команда `\sloppy`: она устанавливает `\tolerance` = 9999, а не 10000 (так что сколь угодно разреженные строки все-таки не допускаются) и при этом задает значение `\emergencystretch`, равное `3em` (так что при необходимости растянуть строки TeX может равномерно распределить дополнительную растяжимость по всему абзацу).

Увеличить значение `\tolerance` можно «глобально», во всем документе, дав в преамбуле команду наподобие

```
\tolerance=400
```

или же «локально», дав аналогичную команду внутри группы, содержащей данный абзац. В последнем случае не забывайте, что закрывающая группу фигурная скобка должна идти после пустой строки, завершающей абзац (см. выше обсуждение команд `\sloppy` и `\raggedright`).

Как менять длину абзаца. Иногда абзац не помещается на полосу из-за того, что он на строку–другую длиннее, чем нужно, и хочется его укоротить. Команда

```
\looseness=-1
```

побуждает TeX стараться, чтобы абзац занял на одну строку меньше, чем при оптимальной верстке. Если абзац короткий (скажем, занимает всего две строки), то из этого, конечно, ничего не получится. Если же абзац достаточно длинный, то у TeX'овского алгоритма обычно хватает гибкости, чтобы достигнуть этой цели.

Можно присвоить параметру `\looseness` и значение -2; в этом случае TeX будет стараться сделать абзац короче на две строки (если не выйдет, то хоть на одну, а если и это не выходит, то оставит все как есть). Можно также присваивать `\looseness` положительные значения — в этом случае TeX будет стараться делать абзацы, которые содержат больше строк, нежели оптимальные.

По умолчанию значение параметра `\looseness` равно, естественно, нулю, и по окончании верстки каждого абзаца этот параметр также устанавливается в нуль. Тем самым нет нужды заботиться о том, чтобы значение `\looseness` менялось внутри группы, и бессмысленно присваивать этому параметру какое-то значение в преамбуле (оно забудется после первого же абзаца текста). Для каждого абзаца, для которого это вообще нужно, значение `\looseness` надо устанавливать заново.

Борьба с последней строкой. Нехорошо, когда последняя строка абзаца слишком коротка (например, короче, чем абзацный отступ следующего абзаца). Чтобы бороться с этим, можно использовать те средства, с которыми мы уже знакомы. Если, например, последняя короткая строка представляет собой обрубок слова, завершающего абзац, то можно либо запретить переносы в этом слове, взяв его в `\mbox` (см. с. 121), либо сказать перед этим словом `\linebreak`, либо, если это слово длинное, указать в нем место для переносов в явном виде (с помощью команды `\-`) только в начале (в надежде, что поджаться на меньшее расстояние TEX'у будет легче). Другой вариант — сказать `\looseness=-1` перед пустой строкой, завершающей абзац: если TEX'у удастся сделать абзац на строку короче, то вряд ли завершающая строка разбитого по-новому абзаца будет короткой.

Разумеется, чудес не бывает: эти рецепты могут подействовать, если абзац достаточно длинен, а не состоит из пары строк.

Другой нежелательный эффект возникает, когда длина последней строки абзаца лишь чуть-чуть меньше, чем ширина полосы. В этом случае разумно довести последнюю строку до края. В этом может помочь еще не рассматривавшийся нами параметр `\parfillskip` . Именно, скажите (перед завершающей абзац пустой строкой) `\parfillskip=0pt` , и TEX постарается растянуть последнюю строку (увеличивая промежутки между словами не только в последней строке, но, при необходимости, и в остальных). Если в результате этих действий не случится overfull'а или underfull'а, то все в порядке.

После окончания абзаца прежнее значение параметра `\parfillskip` не восстанавливается, так что менять его надо внутри группы.

Дополнительные тонкости с переносами. Вы можете влиять на частоту переносов в абзацах с помощью параметра `\hyphenpenalty`. По умолчанию его значение равно 50. Если присвоить этому параметру бóльшее значение, то переносов будет меньше. Точнее говоря, если у TEX'а будет возможность выбирать, сделать лишний перенос или же обойтись без него, растянув строку чуть больше,[8] то TEX будет склоняться ко второму варианту тем чаще, чем больше значение `\hyphenpenalty`. Максимально возможное значение параметра `\hyphenpenalty` равно 10000. Если в момент верстки абзаца это значение именно таково, то переносы в этом абзаце вообще запрещены. Такой режим разумно, например, установить для абзацев, написанных на языке, для которого в вашей реализации TEX'а нет таблицы переносов, чтобы TEX не сделал переносов во французском тексте по английским правилам.

Наряду с параметром `\hyphenpenalty` (отвечающим как за автоматически вставленные переносы, так и за переносы, возможные места для которых вы отметили с помощью команды `\-`), есть и параметр `\exhyphenpenalty`, отвечающий за переносы в словах с дефисом или командой `\slash`. Напомним, что в таких словах автоматический перенос возможен только в том месте, где дефис (или `\slash`) делит слово на части. Так вот, чем больше значение `\exhyphenpenalty`, тем с меньшей охотой TEX будет делать переносы в этих местах. Если же значение `\exhyphenpenalty` равно 10000, то такие переносы будут и вовсе запрещены.

Значение двух описанных выше параметров используется TEX'ом в тот момент, когда он видит пустую строку, завершающую абзац. Соответственно, если вы присваиваете этим параметрам новые значения внутри группы, то группа не должна завершаться до этой пустой строки. Учтите также,

[8] Не превышая значения `\tolerance`, разумеется!

что если вы увеличиваете значение `\hyphenpenalty` и тем самым затрудняете TeX'у переносы слов, то вам может понадобиться увеличить и `\tolerance` или `\emergencystretch`, чтобы он смог побольше растягивать строки.

Полиграфические правила не допускают, чтобы в абзаце шло подряд много (скажем, более трех) строк с переносами. Бороться с таким недостатком помогает параметр `\doublehyphendemerits`: чем его значение больше, тем менее выгодны для TeX'а будут такие последовательности строк, и тем более настойчиво он будет их избегать при переборе вариантов разбиения абзаца на строки. По умолчанию значение этого параметра равно 10000; если переносы идут подряд, можно увеличить значение этого параметра, скажем, до миллиона (такое большое число выбрано не случайно: чтобы этот параметр оказал действие, его значение должно быть того же порядка, что и квадрат встречающихся при переборе возможных разбиений значений badness); если вы делаете это увеличение не в преамбуле документа, а в группе, то, как водится, нужно, чтобы эта группа содержала и пустую строку, завершающую абзац.

Есть также аналогичный параметр `\finalhyphendemerits`: чем больше его значение, тем с меньшей охотой TeX будет делать перенос в предпоследней строке абзаца. Значение этого параметра по умолчанию равно 5000.

Наконец, вот заключительная хитрость. Если вы присвоите значение 0 параметру `\uchyph`, написав

```
\uchyph=0
```

то TeX никогда не будет делать переносов в словах, начинающихся с прописной буквы. Такой режим полезен, например, в том случае, если вы не хотите делать переносы в именах собственных. Чтобы снова разрешить TeX'у переносить слова, начинающиеся с прописной буквы, присвойте параметру `\uchyph` значение 1.

7. Специальные абзацы

В разделе, посвященном абзацам, уже упоминалось о том, как можно печатать текст без выравнивания по правому краю. Сейчас речь пойдет о других подобных случаях, когда требуются абзацы специального вида. Большинство описываемых в этом разделе способов верстки реализовано в виде окружений (см. с. 24); не забывайте, что всякое окружение ограничивает группу, так что если вам нужно будет не только получить текст специального вида, но и сменить шрифт, вы можете дать команду переключения шрифта (наподобие `\itshape`) внутри окружения и не заботиться специально о восстановлении прежнего шрифта: вместе с окружением кончится и группа, и прежний шрифт восстановится автоматически.

7.1. Цитаты

Если вам нужно включить в текст цитату, пример, предупреждение и т. п., то удобно воспользоваться окружением `quote`. Это окружение набирает текст, отодвинутый от краев (полиграфист сказал бы: «втянутый»). Пример:

Каждый сознательный гражданин должен понимать, что

> весьма небезопасно содержать экзотических животных в городской квартире.

Поэтому подумайте, прежде чем покупать на рынке крокодила.

```
Каждый сознательный гражданин
должен понимать, что
\begin{quote}
весьма небезопасно содержать
экзотических животных
в городской квартире.
\end{quote}
Поэтому подумайте, прежде
чем покупать на рынке
крокодила.
```

Как вы можете заметить из этого примера, текст, оформленный окружением `quote`, не имеет абзацного отступа и отделяется от окружающего текста вертикальными промежутками. Если после `\end{quote}` дальнейший текст следует без пропуска строки, то на печати он начнется с новой строки, но без абзацного отступа (после включения цитаты продолжается прерванный абзац); если после `\end{quote}` пропустить строку, то после цитаты текст будет идти с абзацным отступом (если, разумеется, значение параметра `\parindent` не равно нулю — см. с. 25).

Для длинных цитат, состоящих из нескольких абзацев, можно использовать окружение `quotation`. Оно полностью аналогично `quote`, за тем исключением, что в тексте, оформленном этим окружением, делается абзацный отступ.

7.2. Центрирование, выравнивание текста по краю

Для этих целей используются окружения `center` (для центрирования), а также `flushleft` и `flushright` (для выравнивания по левому и правому краю соответственно).

Внутри каждого из этих окружений можно в принципе набирать и самый обычный текст, стандартным образом разбитый на абзацы с помощью пустых строк, но при этом каждая строка получающегося «абзаца» будет центрирована (для окружения `center`) или выровнена по левому/правому краю (для окружений `flushleft` и `flushright` соответственно).

Окружение `flushleft` по своему действию практически эквивалентно команде `\raggedright` (см. с. 123); но вообще, конечно, все перечисленные окружения нужны не для создания абзацев столь странного вида, а для организации текста в виде последовательности строк, каждая из которых центрирована или сдвинута влево или вправо; для этого достаточно после каждой строки, которую вы хотите центрировать (или сдвинуть к краю), поставить команду \\ (см. с. 122); следующая строка будет принадлежать тому же абзацу, и, стало быть, опять же будет центрирована (сдвинута). А если вы не будете делать разбиения на строки командой \\, то это будет сделано автоматически, с появлением несколько странных абзацев (на с. 25 приведен пример того, что может в этом случае получиться). Примеры:

левый
марш

```
\begin{flushleft}
левый\\
марш
\end{flushleft}
```

наше дело
правое

```
\begin{flushright}
наше дело\\
правое
\end{flushright}
```

а вот мы позиционируемся
в центристской части
политического спектра

```
\begin{center}
а вот мы позиционируемся\\
в центристской части\\
политического спектра
\end{center}
```

Как и в случае с `quote` и `quotation`, если после команды `\end`, закрывающей любое из этих окружений, в исходном тексте идет пустая строка, то следующий абзац набирается LaTeX'ом с обычным абзацным отступом, в противном случае — без отступа.

7.3. Стихи

В принципе стихи можно набирать с помощью окружений `flushleft` или `center`, разделяя строки командой \\, а строфы, например, пустой строкой и командой `\smallskip` (см. с. 122 и 146 по поводу этих команд). Кроме того, в LaTeX'е для набора стихов предусмотрено специальное окружение `verse`. Строки в нем разделяются командой \\, а строфы — пустой строкой. При этом строки получаются выровненными по левому краю и отодвинутыми от левой границы текста. Если строка окажется слишком длинной, она будет перенесена на следующую строку (не исключено, что в каких-то словах будет сделан перенос) и сдвинута примерно на 15 пунктов вправо. Пример:

Здесь любит медведь
Иногда посидеть
И подумать: “А чем бы
такое заняться?”

```
\begin{verse}
Здесь любит медведь\\
Иногда посидеть\\ И подумать:
``А чем бы такое заняться?''
\end{verse}
```

Создатель LaTeX'а Лесли Лэмпорт предсказывал, что окружение `verse` будет обругано поэтами.

Наличие или отсутствие абзацного отступа в абзаце после окружения `verse` определяется по тем же правилам, что для `quote` и `quotation`.

7.4. Перечни

Для печати перечней используются окружения `itemize` (для простейших перечней), `enumerate` (для нумерованных перечней) и `description` (для перечней, в которых каждый пункт имеет заголовок — например, словарных статей или иных описаний). В любом случае элементы перечня вводятся командой `\item` (иногда — с необязательным аргументом). Разберем последовательно, как работают указанные окружения.

Простейшие перечни (`itemize`). Каждый элемент перечня вводится командой `\item` без аргумента.

- На печати каждый элемент перечня снабжается темным кружочком («горох» на жаргоне полиграфистов).
- Перечни могут быть вложенными друг в друга:
  - максимальная глубина вложенности равна 4;
  - отступы и символы перед элементами выбираются автоматически.
- На втором уровне элементы перечня отмечаются полужирными короткими тире, на третьем — звездочками, на четвертом — точками.
- При попытке вложить пять таких окружений LaTeX выдаст сообщение об ошибке.

Вот как выглядел в исходном файле предшествующий текст:

```
\begin{itemize}
\item На печати каждый...
\item Перечни могут быть
вложенными друг в друга:
```

```
  \begin{itemize}
  \item максимальная глубина вложенности равна 4;
  \item отступы и символы перед элементами
        выбираются автоматически.
  \end{itemize}
\item На втором уровне элементы...
\item При попытке вложить...
\end{itemize}
```

Внутри окружения `itemize` до первой команды `\item` не должно идти никакого текста или же команд, генерирующих текст. Если вы попытаетесь проигнорировать этот запрет, то LaTeX выдаст вам сообщение об ошибке. Другие команды (например, команды смены шрифта) могут идти и до первого `\item`.

Окружение `itemize` можно использовать также для создания перечней, в которых каждый элемент имеет короткий заголовок. Для создания такого заголовка надо задать команде `\item` необязательный аргумент (в квадратных скобках, как водится). При наличии у этой команды необязательного аргумента стандартный значок, отмечающий элемент перечня («горошина», звездочка и т. п.) не печатается, а вместо него печатается текст, заданный в необязательном аргументе:

- Этот элемент перечня помечен стандартно.

A Здесь мы сами задали заголовок.

999 Здесь тоже.

```
\begin{itemize}
\item Этот элемент перечня
помечен стандартно.
\item[\sffamily A] Здесь мы
сами задали заголовок.
\item[999] Здесь тоже.
\end{itemize}
```

Обратите внимание, что заголовки, заданные нами в необязательных аргументах команд `\item`, печатаются выровненными по правому краю, а также что команды смены шрифта в этих аргументах не распространяются на дальнейший текст.

Если заголовок, заданный вами в необязательном аргументе команды `\item`, будет слишком длинен, то он заедет на левое поле. В таких случаях лучше пользоваться окружением `description`, о котором речь пойдет ниже.

Если первый отличный от пробела символ после команды `\item` является открывающей квадратной скобкой, то LaTeX решит, что эта скобка начинает необязательный аргумент команды `\item`. Если при этом вы использовали эту скобку просто как типографский знак, то в результате получится сообщение об ошибке. Чтобы избежать такой неприятности, надо в этом случае квадратную скобку «спрятать», заключив ее в фигурные скобки:

```
\item {[} --- редко встречающийся знак...
```

Нумерованные перечни (enumerate). В таких перечнях каждый элемент также вводится командой `\item` без аргумента, но на печати он будет отмечен не значком, а номером (эти номера создаются LaTeX'ом автоматически; если вы переставите какие-то элементы перечня, что-то добавите или удалите, нумерация автоматически изменится).

1. В окружении `enumerate` элементы списка нумеруются цифрами или буквами.
2. Нумерация производится автоматически.
3. Перечни могут быть вложенными друг в друга:
   (a) максимальная глубина вложенности равна 4;
   (b) отступы и обозначения для элементов выбираются автоматически.
4. На втором уровне элементы обозначаются строчными буквами, на третьем — римскими цифрами, на четвертом — прописными буквами.
5. При попытке вложить пять таких окружений LaTeX выдаст сообщение об ошибке.

В исходном тексте это выглядело так:

```
\begin{enumerate}
\item В окружении \texttt{enumerate}...
\item Нумерация производится автоматически.
\item Перечни могут быть вложенными друг в друга:
  \begin{enumerate}
  \item максимальная глубина  вложенности равна 4;
  \item отступы и обозначения для элементов...
  \end{enumerate}
\item На втором уровне элементы обозначаются...
\item При попытке вложить пять таких окружений...
\end{enumerate}
```

Внутри окружения `enumerate` до первой команды `\item` не должно идти никакого текста или же команд, генерирующих текст.

На номера элементов нумерованного перечня можно организовать автоматические ссылки с помощью команды `\ref` (см. с. 27). Делается это так.

Представим себе, что вам нужно сослаться на какой-то пункт нумерованного перечня (например, чтобы написать «Согласно пункту 3 настоящих Правил...»). Если вы в ходе работы над текстом переставите какие-то пункты или добавите новые, то номер пункта может измениться. Вместо того чтобы каждый раз отсчитывать, которым по счету идет этот пункт, можно пометить элемент перечня с помощью команды `\label` (см. разд. I.2.11). Команду `\label` лучше ставить сразу после команды `\item`, вводящей помечаемый элемент перечня. но можно поставить ее и позже — до следующего `\item`.

Ссылка на метку производится с помощью команды `\ref`. У нее также один обязательный аргумент — та самая метка, на которую вы хотите сослаться (ссылка на страницу, на которой расположена метка, производится, как обычно, с помощью команды `\pageref`). Пример:

1. Переходите улицу только на зеленый свет.

2. Стоящий трамвай обходить можно, а автобус — нет.

Согласно правилу 2, сформулированному на с. 134, обходить стоящий автобус нельзя.

```
\begin{enumerate}
\item Переходите улицу только
на зеленый свет.
\item \label{tram}
Стоящий трамвай обходить
можно, а автобус~--- нет.
\end{enumerate}
Согласно правилу~\ref{tram},
сформулированному
на с.~\pageref{tram},
обходить стоящий
автобус нельзя.
```

Символы неразрывного пробела мы поставили затем, чтобы номер правила или страницы не остался в одиночестве в начале строки.

В окружении `enumerate` команда `\item` может иметь необязательный аргумент, который работает так же, как в окружении `itemize`. Если первый отличный от пробела символ после `\item` является открывающей квадратной скобкой, необходимо взять эту квадратную скобку в фигурные скобки (как и в случае с окружением `itemize`).

Перечни с заголовками (`description`). В этих перечнях каждый элемент, как уже было сказано, снабжен заголовком. Поэтому элементы перечня вводятся командой `\item` с необязательным аргументом (заключенным, стало быть, в квадратные скобки — см. с. 23), представляющим собой этот заголовок. Пример:

Летом можно собирать различные ягоды:

черника: темно-синие, очень вкусные, хороши в свежем виде, варенье тоже получается хорошее;

голубика: синие, более водянистые, чем черника, и не такие вкусные;

брусника: ярко-красные, из них получается очень вкусное варенье.

```
Летом можно собирать
различные ягоды:
\begin{description}
\item[черника:] темно-синие,
очень вкусные, хороши в
свежем виде, варенье
тоже получается хорошее;
\item[голубика:]
синие, более водянистые,
чем черника, и не такие вкусные;
\item[брусника:] ярко-красные,
из них получается очень
вкусное варенье.
\end{description}
```

Как вы могли заметить, заголовки элементов перечня оформляются в окружении `description` полужирным шрифтом. Если вас не устраивает этот шрифт, можно аргумент команды `\item` начать с команды переключения шрифта, скажем, `\normalfont` или `\slshape`.

Внутри окружения `description` до первой команды `\item` не должно идти никакого текста или же команд, генерирующих текст.

Если в заголовке элемента перечня присутствует закрывающая квадратная скобка, то LaTeX решит, что именно на ней заканчивается необязательный аргумент команды `\item`, в результате чего на печати получится совсем не то, что вы хотели. Чтобы избежать этой неприятности, надо эту квадратную скобку (либо, что еще проще, весь заголовок) заключить дополнительно в фигурные скобки (внутри квадратных).

Другие виды перечней. Если вас не устраивает стандартное оформление перечней (например, вид пометок, которыми отмечаются элементы перечня `itemize`), его несложно изменить. Как это сделать, будет рассказано в разд. VII.3.5. Несколько труднее, к сожалению, сделать так, чтобы буквы, которыми нумеруются элементы нумерованного перечня, были русскими, а не латинскими, как в примерах в этой книге (в гл. IX мы расскажем, как этого добиться). Можно менять оформление перечней и более серьезным образом, создавая перечни иного типа, чем рассмотренные выше. На данный момент наш TeXнический уровень не столь высок, чтобы можно было освоить эти возможности LaTeX'а, но в гл. IX будет рассказано и об этом.

7.5. Буквальное воспроизведение (verbatim, verb)

Окружение `verbatim` предназначено для буквального воспроизведения имеющихся в файле символов (шрифтом типа пишушей машинки). Одной только команды `\ttfamily` для этого недостаточно, поскольку воспроизводимый текст может содержать, например, команды TeX'а, и необходимо, чтобы они печатались, а не исполнялись.

Между `\begin{verbatim}` и `\end{verbatim}` могут идти любые символы (в том числе символ `\` и непарные фигурные скобки), за исключением последовательности символов «`\end{verbatim}`». После этого надо написать `\end{verbatim}` в отдельной строке, ничего, кроме этого текста, не содержащей (для всех прочих LaTeX'овских окружений это не обязательно). При этом между `\end` и `{verbatim}` не должно быть пробела (также вопреки общим правилам: обычно такой пробел, как и вообще пробел после имени команды, состоящего из букв, ни на что не влияет).

Короткие последовательности символов удобно набирать для буквального воспроизведения с помощью команды `\verb`. Непосредственно после `\verb` должен стоять любой символ, не являющийся буквой или звездочкой, далее — воспроизводимый текст (укладывающийся в одну строку), не содержащий того символа, который стоял непосредственно после `\verb`, а затем — снова тот символ, что стоял непосредственно после `\verb`. После `\verb` не должно быть пробела. Пример:

Команда `\dots` задает многоточие. Знак `"` в TeX'е используется редко.

```
Команда \verb"\dots" задает
многоточие. Знак~\verb|"|
в \TeX'е используется
редко.
```

Описанные окружение и команда удобны, когда надо имитировать машинописный текст, текст на мониторе компьютера, или набирать тексты компьютерных программ. В данном руководстве `\verb` и `verbatim` широко использовались для набора LaTeX'овских и TeX'овских команд.

У команды `\verb` и окружения `verbatim` есть варианты «со звездочкой» (см. с. 25). От своих вариантов без звездочки они отличаются тем, что пробел изображается знаком ␣.

Команду `\verb` и окружение `verbatim` нельзя использовать в сносках; если вам необходимо напечатать в сноске что-нибудь вроде `\sqrt`, то придется делать это вручную, с помощью команды `\symbol`: `\texttt{\symbol{'\\}sqrt}` или `\texttt{\symbol{"5C}sqrt}`.

Если вы забудете «закрывающий символ» в команде `\verb` или сделаете опечатку в тексте `\end{verbatim}`, то в лучшем случае получите уйму сообщений об ошибке, а в худшем — завесите компьютер.

Если вы воспроизводите в режиме `verbatim` текст, простирающийся на многие страницы (например, компьютерную программу), то TeX'у может не хватить памяти. Чтобы избежать такой неприятности, надо или распределить воспроизводимый текст по нескольким окружениям `verbatim`, или подключить стилевой пакет `verbatim`, после чего можно будет спокойно задавать сколь угодно длинные окружения `verbatim` и `verbatim*` (только не забудьте про `\end{verbatim}` в конце). Кроме того, при подключении этого пакета становится доступной команда `\verbatiminput`, позволяющая дословно воспроизвести содержимое произвольного текстового файла: именно, если вы напишете `\verbatiminput{something.txt}`, то это будет равносильно тому, как если бы вы написали

```
\begin{verbatim}
⟨содержимое файла something.txt⟩
\end{verbatim}
```

7.6. Абзацы нестандартной формы

Пусть нам потребовалось создать абзац с «отрицательным» абзацным отступом, в котором все строки, кроме первой, начинаются на расстоянии одного сантиметра от полей. Этого можно добиться следующим образом:

Отрицательный абзацный отступ (по-английски hanging indentation).

```
\hangindent=1cm \noindent
Отрицательный абзацный отступ
(по-английски hanging indentation).
```

Здесь TeX'овский параметр `\hangindent` означает величину отступа от полей во всех строках абзаца, кроме первой (по умолчанию значение этого параметра равно нулю). Обратите внимание, что мы начали абзац командой `\noindent`, чтобы первая строка не началась с абзацным отступом (см. разд. 9.2).

Пусть теперь нам хочется, чтобы дополнительный отступ, величина которого задана параметром `\hangindent`, начинался не со второй строки, а, скажем, с третьей. Для этого надо установить еще один TeX'овский параметр, обозначаемый `\hangafter`:

Можно сделать так, чтобы отступ начался не с первой строки, а там, где нам это потребовалось.

```
\hangindent=1cm \hangafter=2
\noindent Можно сделать
так, чтобы отступ начался не с
первой строки, а там, где нам
это потребовалось.
```

Значение параметра `\hangafter` — номер строки, после которой начинается дополнительный отступ. По умолчанию значение `\hangafter` равно единице (как и было в нашем первом примере).

Можно также добиться того, чтоб дополнительный отступ не начинался после какой-то строки, а напротив, присутствовал только в нескольких первых строках абзаца. Для этого надо присвоить параметру `\hangafter` отрицательное значение: если величина `\hangafter` равна $n < 0$, то дополнительный отступ, равный `\hangindent`, будет присутствовать в строках номер $1, 2, \ldots, |n|$. Пример:

С помощью рассмотренных нами средств TeX'а можно выкапывать в абзацах небольшие ямки. На место образующегося белого прямоугольника можно, например, вклеить иллюстрацию.

```
\hangindent=1.5cm
\hangafter=-3 \noindent
С помощью рассмотренных нами
средств \TeX'а можно
выкапывать в абзацах небольшие
ямки. На место образующегося
белого прямоугольника можно,
например, вклеить иллюстрацию.
```

Если значение параметра `\hangindent` отрицательно и равно h, то дополнительный отступ размером $|h|$ будет отсчитываться от правого, а не левого поля (в каких именно строках будет этот дополнительный отступ, по-прежнему определяется значением `\hangafter`):

На сей раз нам захотелось приклеить картинку не слева, а справа. Что ж, TeX позволяет сделать и так, было бы желание. Вскоре вы сможете убедиться, что и это — не предел.

```
\hangindent=-2cm \hangafter=2
\noindent
На сей раз нам захотелось
приклеить картинку не слева,
а справа. Что ж, \TeX\ позволяет
сделать и так, было бы желание.
Вскоре вы сможете убедиться,
что и это~--- не предел.
```

После каждой команды «завершить абзац» (иными словами, после каждой пустой строки или команды `\par`) восстанавливаются принятые по умолчанию значения параметров `\hangindent` и `\hangafter`. Отметим еще, что не следует менять эти параметры внутри LaTeX'овских окружений наподобие `itemize` или `quote`: в таких окружениях LaTeX устанавливает эти параметры самостоятельно, и их ручная переустановка может привести к непредсказуемым результатам.

Если вам не хватает возможностей,
приведенных выше, то посмотрите, как можно с помощью TeX'а создать абзац совсем уж причудливой формы. Все переносы в словах и места разрывов строк были найдены TeX'ом автоматически. Получившийся абзац напоминает автору изображение шахматной ладьи. (Предложение в скобках пришлось дописать, чтобы изображение не выглядело незавершенным.)

Начало этого причудливого абзаца выглядело в исходном тексте так:

```
\parshape=14
0cm 6cm  .1cm 5.8cm  .17cm 5.66cm  .5cm 5cm
.9cm 4.2cm  1.05cm 3.9cm  1.1cm 3.8cm  1.1cm 3.8cm
1.05cm 3.9cm  .9cm 4.2cm  .5cm 5cm .17cm 5.66cm
.1cm 5.8cm  0cm 6cm
\noindent \small
Если вам не хватает возможностей...
```

Смысл этого текста следующий. Число 14, следующее непосредственно после `\parshape` и знака равенства, задает количество строк, имеющих нестандартные длину и/или отступ от левого поля. После этого числа, через пробел (конец строки, как мы помним, — тоже пробел), перечислены отступы от левого поля и длины строк: `0cm` — отступ первой строки от левого поля, `6cm` — ее длина, `.1cm` — отступ второй строки от левого поля, `5.8cm` — ее длина, и т. д. Если написано, что `\parshape` равно n, то после этого должно следовать $2n$ длин. Если реально в абзаце получится менее n строк, то указания на длину и отступ отсутствующих строк будут проигнорированы TeX'ом; если строк получается больше, чем n, то все последующие строки будут иметь те же отступ и длину, что заданы для строки номер n. Заметим, наконец, что абзац мы начали командой `\noindent`, чтобы отступ самой первой строки был действительно равен нулю (если абзац начинается без `\noindent`, то в первой строке будет еще присутствовать пробел длиной в `\parindent`).

После пустой строки или команды `\par` действие параметров, заданных командой `\parshape`, прекращается.

У абзаца, форма которого задана с помощью `\hangindent` или `\parshape`, длина и отступ строки зависят, как вы могли заметить, от ее номера. Если такой абзац содержит выключную формулу, то TeX считает, что эта формула занимает три строки, причем сама формула расположена в средней из этих трех (реально формула может, разумеется, занимать больше места).

8. Сноски

Чтобы сделать сноску к какому-то месту в тексте, достаточно использовать команду `\footnote` с одним обязательным аргументом — текстом сноски. В стандартных классах LaTeX'а сноски[9] нумеруются подряд на протяжении всей главы или даже (в классе `article`) всего документа. В исходном тексте предыдущий фрагмент выглядел так:

```
сноски\footnote{Вроде этой.} нумеруются ...
```

В разделе VII.3, посвященном «счетчикам», мы расскажем о том, какие возможности есть для того, чтобы помечать сноски по-другому.

Если после слова, к которому делается сноска, должен стоять знак препинания, то в исходном тексте его надо поставить *после* закрывающей фигурной скобки, ограничивающей аргумент команды `\footnote`.

Текст сноски может состоять из нескольких абзацев; в этом случае они, как обычно, разделяются пустой строкой.

К сожалению, весьма непросто заставить TeX автоматически нумеровать сноски так, чтобы нумерация начиналась заново на каждой странице. В LaTeX'е, в частности, такая возможность не предусмотрена. Если вы готовы пожертвовать автоматической нумерацией сносок, то можно воспользоваться командой `\footnote` с необязательным аргументом. Этот необязательный аргумент ставится (в квадратных скобках) перед обязательным[2003]. Предыдущий фрагмент выглядел в исходном тексте так:

```
обязательным\footnote[2003]{Вот какой ... }
```

При использовании команды `\footnote` с необязательным аргументом автоматическая нумерация сносок не сбивается: предыдущая сноска имела номер 9, затем мы искусственно создали сноску номер 2003, а следующая сноска[10] будет иметь номер 10.

В случае, если вы хотите сделать сноску к тексту, входящему в «блок» (например, в аргумент команды `\mbox`; в гл. VIII мы расскажем о том, что такое блок в общем случае и какими командами блоки генерируются), команда `\footnote` непригодна. Вот как надо ставить сноски в этом случае:

---

[9] Вроде этой.

[2003] Вот какой интересный номер!

[10] Вот эта.

```
Роман \mbox{‘‘Три\footnotemark\
мушкетера’’}\footnotetext{А не четыре!} написал Дюма.
```

В этом случае получится нормальная сноска, напоминающая нам, что в названии романа “Три[11] мушкетера” фигурируют три мушкетера, а не четыре. Если бы мы просто написали `\footnote`, то увидели бы на печати только номер, но не саму сноску. Обратите также внимание на «backslash с пробелом» после команды `\footnotemark`: мы его поставили, чтобы между словами «три» и «мушкетера» на печати был пробел (см. с. 20).

Если вы к тому же хотите вручную задать номер сноски к тексту, входящему в «блок», то нужно задать этот номер дважды: первый раз в качестве необязательного аргумента команды `\footnotemark` (обязательных аргументов у этой команды не предусмотрено), а второй раз — в качестве необязательного аргумента команды `\footnotetext` (необязательный аргумент этой команды должен идти *перед* обязательным):

```
Роман \mbox{‘‘Три\footnotemark[99]
мушкетера’’}\footnotetext[99]{А не четыре!}
написал Дюма.
```

9. Между абзацами

В предыдущих разделах мы обсуждали, что происходит с документом «на уровне строки». Теперь изменим масштаб наших рассмотрений: будем смотреть не только на строки и абзацы, но и на то, как они расположены на странице.

9.1. Понятие о режимах TEX’а

Несколько упрощая ситуацию, можно сказать, что в процессе обработки исходного текста TEX в каждый момент находится в одном из трех режимов: горизонтальном, вертикальном или математическом.

- В процессе обработки текста (от появления первой же буквы до команды «закончить абзац», например, пустой строки) TEX находится в горизонтальном режиме.
- Между абзацами, а также в начале работы (например, в процессе обработки преамбулы к LATEX’овскому файлу) TEX находится в вертикальном режиме.

[11] А не четыре!

- При обработке математических формул (см. гл. II) TeX находится в математическом режиме.

В вертикальном режиме все пробелы и пустые строки игнорируются, так что между пустой строкой, завершающей абзац, и новой порцией собственно текста незачем заботиться о лишних или недостающих пробелах (ср. с. 19).

В качестве команды «закончить абзац» можно использовать, наряду с известной вам пустой строкой, команду `\par`:

Это — абзац, который мы не намерены завершать пустой строкой, как раньше. А это другой абзац.	`Это~--- абзац, который` `мы не намерены завершать` `пустой строкой, как` `раньше.\par А это` `другой абзац.`

Иногда для ясности, когда в исходном файле присутствует сложная комбинация из TeX'овских команд, имеет смысл обозначить конец абзаца именно таким способом.

Идущие подряд несколько команд `\par`, команда `\par`, за или перед которой следует пустая строка, и т. п. — все это равносильно одной пустой строке или одной команде `\par` (точно так же, как несколько пустых строк равносильны одной); дополнительный промежуток между абзацами вы таким образом не создадите. В разд. 9.4 рассказывается, как получить на печати дополнительные вертикальные промежутки.

Сказанное в предыдущем абзаце можно с помощью понятия режима сформулировать так: в вертикальном режиме команда `\par` ничего не делает.

9.2. Подавление абзацного отступа

Иногда возникает необходимость создать абзац, в котором нет абзацного отступа. Для этой цели удобно воспользоваться командой `\noindent`. В том абзаце, отступ в котором вы хотите подавить, эта команда должна идти первой (до любого текста):

В этом абзаце отступа не будет. В этом абзаце отступ будет присутствовать.	`\noindent В этом абзаце` `отступа не будет.\par` `В этом абзаце отступ` `будет \noindent` `присутствовать.`

Команда `\noindent` действует только на тот абзац, который с нее начинается; если ее поместить внутри абзаца, то вообще ничего не произойдет (что и иллюстрирует второй из абзацев в нашем примере). Стало быть, между `\noindent` и абзацем, к которому она относится, не должно быть пустой строки (иначе получится, что `\noindent` относится к «пустому абзацу», заканчивающемуся этой пустой строкой).

В большинстве случаев, когда разумно сделать абзац без отступа, LaTeX заботится об этом сам, так что вам не придется пользоваться командой `\noindent` чересчур часто.

Пользуясь понятием режима, можно сказать так: в вертикальном режиме команда `\noindent` означает «начать новый абзац без абзацного отступа», а в горизонтальном (и математическом, коль на то пошло) режиме она означает «ничего не делать».

9.3. Управление разрывами страниц

Как вы могли убедиться, TeX предоставляет широкие возможности для управления видом абзаца, местами разрывов строк и т. п. С разрывами страниц все обстоит не столь хорошо. Дело в том, что при верстке абзаца TeX сначала читает его целиком, а затем перебирает различные способы разбиения на строки и выбирает из них оптимальный. При разбиении на страницы такой подход невозможен: если читать сразу весь текст, а затем перебирать различные варианты разбиения его на страницы, то компьютеру не хватит памяти. Поэтому разбиение на страницы в TeX'е — процесс «одноразовый». Обработав очередной абзац, TeX проверяет, набралось ли уже достаточно строк, чтоб заполнить страницу. Если оказывается, что достаточно, он производит разрыв страницы, и при этом выбор обычно невелик (часто бывает возможно сместить место разрыва страницы на строчку–другую за счет того, что некоторые интервалы между строками можно слегка растягивать или сжимать; таковы обычно интервалы между абзацами, между текстом и выключными формулами, но не между строками внутри абзаца). Имея все это в виду, рассмотрим, какие команды предоставляет LaTeX для управления разрывами страниц.

Запрет разрыва страницы. Чтобы запретить разрыв страницы, используется команда `\nopagebreak`. Если поставить ее после конца абзаца, то разрыв страницы после этого абзаца будет запрещен. Если после конца абзаца присутствуют совместно как команда `\nopagebreak`, так и команда для дополнительных вертикальных промежутков, то команда `\nopagebreak` должна идти первой, в противном случае она не подействует.

Команда `\nopagebreak` может принимать необязательный аргумент — целое число от 0 до 4. Будучи снабжена этим аргументом, она не запрещает разрыв страницы в указанном месте, но делает его менее выгодным с точки зрения TEX'a (тем менее выгодным, чем больше аргумент). Команда `\nopagebreak[4]` означает полный запрет разрыва, как если бы команда была дана вообще без аргумента. Если аргумент равен 0, это означает только, что в данном месте страницу в принципе *можно* разорвать.

Принудительный разрыв страницы. Для принудительного разрыва страниц в LATEX'е существует несколько способов. Первый и самый простой — команда `\newpage`. Под действием этой команды текущая страница завершается и дополняется снизу пустым пространством, если высота страницы получается меньше, чем надо.

Команда `\clearpage` также предназначена для принудительного разрыва страницы. Если пользоваться только теми средствами LATEX'a, которые были описаны до этого момента в нашей книге, то она будет работать в точности так же, как `\newpage`. В том же случае, если к моменту подачи этой команды остались так называемые «плавающие» иллюстрации или таблицы (см. разд. IV.8), то перед выдачей новой страницы они будут, скорее всего, напечатаны.

Команда `\cleardoublepage` делает то же, что и `\clearpage`, но при этом в некоторых классах документов (в тех, которые предусматривают разное оформление страниц с четными и нечетными номерами — см. разд. IV.1 по поводу классовой опции `twoside`) новая страница обязательно имеет нечетный номер (если необходимо, при этом создается дополнительная пустая страница).

Если поставить подряд две команды `\newpage` (или `\clearpage`), то в печатном тексте чистая страница не получится. Чтобы создать чистую страницу, надо LATEX немного обмануть: между двумя командами для разрыва страницы дать команду `\mbox{}`.

Наконец, существует команда `\pagebreak`, формально аналогичная команде `\linebreak` (см. с. 122). Если дать ее без аргументов, то страница в этом месте будет разорвана; при этом не исключено, что будет сделана попытка выровнять ее по высоте с остальными страницами за счет растяжения тех вертикальных интервалов, которые можно растянуть — как правило, это интервалы между абзацами. (Команда `\newpage` такой попытки не делает.) Если дать команду `\pagebreak` с необязательным аргументом (целым числом от 0 до 4), то этот аргумент будет выражать степень желательности разрыва страницы в данном месте: если 0, то это всего лишь разрешение разорвать страницу, если 4, то разрыв обязателен, в остальных случаях степень желательности растет с ростом аргумента от 1 до 3.

Каждую из названных команд можно дать не только между абзацами, но и внутри абзаца; при этом разрыв страницы произойдет (или будет запрещен) после той строки, в которую попадает текст, соседствующий с этой командой.

Изменение высоты отдельной страницы. Иногда при окончательной отделке текста бывает необходимо немного увеличить или уменьшить размер отдельно взятой страницы (чтобы, например, втиснуть в нее еще одну строку, которая иначе окажется в одиночестве на следующей полосе). Для этого есть следующее средство: на той странице, размер которой надо увеличить на одну строку, поместить между абзацами команду

```
\enlargethispage{\baselineskip}
```

Если надо увеличить размер на две строки, а не на одну, напишите в фигурных скобках `2\baselineskip` вместо `\baselineskip`; можно также в аргументе команды `\enlargethispage` написать `-\baselineskip`, `-2\baselineskip`, и т. п. В этом случае высота полосы уменьшится на одну, две и т. д. строки.

Добавим несколько TEX'нических подробностей. Во-первых, если текст набирается в две колонки, то команда `\enlargethispage` действует только на одну из них — на ту, в которую она попала. Во-вторых, при действии команды `\enlargethispage` увеличенная полоса может наложиться на строку с колонцифрой, если таковая предусмотрена стилем оформления документа. И наконец, в аргументе команды `\enlargethispage` может стоять не только кратное `\baselineskip`, но и любая длина, выраженная в TEX'овских единицах (скажем, `5mm`).

Подробности о параметре `\baselineskip` и колонцифре — в следующей главе. О двухколонном наборе будет рассказано в разд. 9.6.

Висячие строки. Вообще говоря, не следует допускать, чтобы на страницу попадала только первая или только последняя строка абзаца. В TEX'е предусмотрены два параметра, влияющие на вероятность появления разрывов страницы в этих местах. Именно, параметр `\clubpenalty` определяет нежелательность разрыва страницы после первой строки абзаца, а `\widowpenalty` — перед последней. Чем выше значение этих параметров, тем с меньшей охотой TEX будет допускать такие разрывы (если, конечно, есть возможность выбора); значение 10000 означает, что разрыв полностью запрещен. По умолчанию и `\clubpenalty`, и `\widowpenalty` равны 150.

9.4. Вертикальные промежутки

Большинство вертикальных промежутков (например, между заголовком раздела и его текстом) LATEX устанавливает самостоятельно, и вам об этом можно не заботиться. Иногда возникает необходимость сделать дополнительный вертикальный промежуток между абзацами. Как вы помните, как внутри абзацев для задания промежутков вручную разумнее пользоваться не командами, явно задающими размер промежутка, а командами вроде `\,` или `\quad`; аналогичным образом, для задания промежутков между абзацами рекомендуются такие команды:

- `\smallskip` задает такой промежуток;
- `\medskip` задает такой промежуток;
- `\bigskip` задает такой промежуток.

Проще всего поставить эти команды непосредственно после пустой строки или команды `\par`, завершающей абзац:

После этого абзаца мы оставим дополнительный пробел.

А теперь начнем новый абзац.

```
После этого абзаца мы оставим
дополнительный пробел.
\par\smallskip
А теперь начнем новый
абзац.
```

Конкретная величина промежутков, задаваемых этими командами, зависит от класса документа. Эти размеры совпадают со значениями параметров `\smallskipamount`...`\bigskipamount`, о которых шла речь на с. 122.

Если вы хотите задать размер вертикального промежутка в явном виде, можно воспользоваться командой `\vspace`. Подобно команде `\hspace` (см. с. 106), у нее есть один обязательный аргумент — величина промежутка. Например, можно написать

```
\vspace{2ex}
```

Команду `\vspace` удобнее всего ставить после конца абзаца (подобно таким командам, как `\smallskip`).

Можно поставить команду `\vspace` (или `\smallskip` и т. п.) не после пустой строки или `\par`, а непосредственно перед ними, после всего текста абзаца. Если поставить какую-либо из этих команд внутри абзаца, то дополнительный вертикальный пробел получится не между абзацами, а между строками абзаца.

Если дать команду `\vspace` сразу же после `\newpage` или `\clearpage`, то вертикального отступа в начале новой страницы *не получится*; вертикальный отступ, создаваемый `\vspace`, пропадет и в том случае, если он оказывается в начале новой страницы, получившейся «естественным образом». Чтобы вертикальный отступ в начале страницы не пропадал, надо воспользоваться вариантом со звездочкой после имени команды:

если написать `\vspace*{1cm}`, то будет создан вертикальный промежуток в `1cm`, не пропадающий даже в том случае, если команда дана сразу после `\newpage` или `\clearpage` или в этом месте произошел разрыв страницы.

Можно заставить команду `\vspace` создать промежуток не фиксированной, а переменной длины. Именно, в самом общем виде эта команда записывается так:

`\vspace{`x `plus` y `minus` z`}`

Здесь x, y и z — длины, выраженные в TEX'овских единицах, а `plus` и `minus` — так называемые «ключевые слова» TEX'а (в отличие от команд, перед ними *не надо* ставить backslash). При этом x обозначает «естественную» величину отступа: если при верстке страницы вертикальные интервалы не приходится растягивать или сжимать (например, в случае, когда, мы разрешили TEX'у оставлять внизу страницы пустое место; в дальнейшем мы обсудим, как это делать), то будет сделан пробел размером ровно x. При необходимости, однако (например, ради того, чтобы все страницы имели одинаковую высоту), этот интервал можно будет и изменить: y указывает степень растяжимости, а z — степень сжимаемости интервала. Говоря TEXническим языком, команда `\vspace` вставляет в страницу «клей»[12]; расстояния, указанные после `plus` и `minus`, называются соответственно `plus`- и `minus`-компонентами этого клея. Если `plus`- или `minus`-компонента в аргументе команды `\vspace` не указана, то соответствующий интервал не сможет растягиваться (сжиматься). Большинство вертикальных интервалов, автоматически вставляемых LATEX'ом, обладают растяжимостью и/или сжимаемостью, что помогает при нахождении оптимальных разрывов страниц.

Один частный случай растяжимых промежутков настолько важен, что в LATEX'е для него предусмотрена специальная команда. Именно, в аргументе `\vspace` или `\vspace*` можно вместо длины, заданной в TEX'овских единицах, написать `\fill`. Это задает промежуток нулевого размера, но обладающий способностью бесконечно растягиваться. Если, например, написать

```
\clearpage\vspace*{\fill}
\begin{center}
Заголовок
\end{center}
\vspace*{\fill}\clearpage
```

то слово «заголовок» будет расположено точно по центру отдельной страницы, созданной командами `\clearpage`.

Перед командой `\fill` в аргументе `\vspace` или `\vspace*` можно поставить коэффициент — целое число или десятичную дробь, и тогда растяжимость умножится на этот коэффициент. Например, если написать

```
\clearpage\vspace*{0.5\fill}
```

[12]Свойства которого мало похожи на свойства настоящего клея. См. гл. VIII.

```
\begin{center}
Заголовок
\end{center}
\vspace*{\fill}\clearpage
```

то перед словом «заголовок» будет оставлено ровно в два раза меньше места, чем после него, так как `0.5\fill` растяжим в два раза меньше, чем `\fill`.

Теперь можно признаться, что горизонтальные промежутки, создаваемые командой `\hspace`, также могут быть растяжимыми; чтобы этого добиться, надо задать в аргументе команды `\hspace` не только «естественную длину», но еще и `plus`- и/или `minus`-компоненту. Например, если сказать

```
\hspace{1cm plus 2mm minus 1em}
```

то при верстке абзаца соответствующий интервал сможет растягиваться или сжиматься. Можно также, вместо длин с `plus`- или `minus`-компонентами, написать `\fill` (возможно, с коэффициентом). В простых приложениях такие конструкции, как правило, не встречаются. Мы еще будем говорить о них в разд. VIII.3.3.

9.5. Интерлиньяж

В полиграфии этим красивым словом называется интервал между строками. Команды наподобие `\small`, устанавливающие размер шрифта, автоматически устанавливают и размер интервала между строками, так что вручную менять его не следует (потому мы и не рассказываем, как это делать; любопытствующий читатель может узнать некоторые подробности в приложении В и все подробности в книге [2]). Можно, однако (и иногда это бывает необходимо), *пропорционально* увеличить или уменьшить все интервалы между строками — например, чтобы подогнать число полос в документе к требуемому. Если, скажем, вы хотите увеличить интервалы между строками на 1%, т. е. в 1.01 раза, то в преамбуле следует написать так:

```
\renewcommand{\baselinestretch}{1.01}
```

Если нужно пропорционально увеличить или уменьшить интерлиньяж в каком-то фрагменте текста, то можно написать так:

```
{% Открывающая фигурная скобка небходима
\renewcommand{\baselinestretch}{1.01}
\selectfont
Текст, в котором надо изменить интерлиньяж...
....
Конец этого текста

}% Закрывающая скобка, парная к открывающей
```

(если опять надо увеличить на 1%; в других случаях — соответственно). Не забудьте, что закрывающая фигурная скобка должна стоять *после* пустой строки, завершающей последний абзац, иначе ничего не выйдет (см. с. 120).

Почему этот прием работает и каков смысл команды `\selectfont`, объясняется в приложении В.

Между абзацами можно организовать дополнительные вертикальные интервалы. Именно, в TEX'е есть параметр `\parskip` со значением длины; если присвоить ему ненулевое значение, например, написав

```
\parskip=3mm
```

то между абзацами будет делаться отступ в `3mm` (в дополнение к обычному межстрочному интервалу). Без особой необходимости не следует присваивать параметру `\parskip` новое значение, поскольку оно вполне разумно устанавливается в стандартных LATEX'овских классах.

На самом деле в стандартных классах `\parskip` является *растяжимой* длиной (см. с. 147). Именно, естественный размер `\parskip` равен нулю, но у него есть еще `plus`-компонента, равная одному пункту. Стало быть, если вертикальные интервалы на странице не варьируются, то никакого дополнительного интервала между абзацами не делается, но если страницу при верстке приходится растягивать по вертикали, то каждый из интервалов между абзацами может быть растянут. При желании можно изменить как естественный размер, так и растяжимую компоненту параметра `\parskip` с помощью команды `\setlength`, о которой пойдет речь в разд. VII.4.

9.6. Набор в две колонки

Если вам необходимо набирать в две колонки весь документ, то это надо сделать, указав в команде `\documentclass` соответствующую «классовую опцию» (см. разд. IV.1). Если же в две колонки надо набрать не весь текст, а только его часть, к вашим услугам команда `\twocolumn`. Действует она так: сначала выполняется команда `\clearpage`, а затем с новой страницы, созданной этой командой, начинается набор в две колонки.

Если вы хотите, чтобы колонки на печати были разделены вертикальной линейкой, присвойте ненулевое значение параметру под названием `\columnseprule` (его значение равно ширине линейки). По умолчанию значение этого параметра равно `0pt`, так что линейка не печатается; хорошая линейка получается при значении `0.4pt`.

К сожалению, на последней странице двухколонного набора высоты колонок не выравниваются; см. ниже по поводу того, что можно с этим сделать.

Иногда бывает необходимо в начале новой страницы поместить один или несколько абзацев текста во всю ширину страницы, а оставшийся текст на этой странице набрать в две колонки. Для этих целей можно использовать команду `\twocolumn` с необязательным аргументом. Необязательный аргумент (в квадратных скобках, как водится) — это тот текст, который будет напечатан во всю ширину страницы; если он состоит из нескольких абзацев, то абзацы, как обычно, разделяются пустыми строками.

Команда `\onecolumn` осуществляет переход от двухколонного набора к одноколонному (предварительно она опять-таки выполняет команду `\clearpage`).

Если вам хочется, чтобы колонки на последней странице были одинаковой высоты, чтобы можно было начинать двухколонный набор не только с новой страницы, а также если вам нужны не две колонки, а три или больше, можно подключить стилевой пакет `multicol`. В этом случае к вашим услугам окажется окружение `multicols`.

Единственный обязательный аргумент окружения `multicols` — целое число, равное числу колонок. Если, скажем, вам нужно, начиная с какого-то места, начать набор в три колонки, скажите

```
\begin{multicols}{3}
Текст
\end{multicols}
```

Набор в три колоноки начнется с того места на странице, куда попало `\begin{multicols}`, колонки на последней странице трехколонного набора будут выровнены по высоте. Присвоив ненулевое значение параметру `\columnseprule`, можно разделить колонки линейками.

9.7. Заключительные замечания о разрывах страниц и вертикальных интервалах

Мы уже отмечали, что TeX'овские алгоритмы создания страниц не обладают той же гибкостью, что алгоритм разбиения абзаца на строки. Поэтому не надо слишком увлекаться принудительными разрывами и запретами разрывов страниц и командами наподобие `\vspace*`. Даже такая замечательная программа, как TeX, не сможет удовлетворить логически противоречивым требованиям; если ограничений на разрывы страниц слишком много, то TeX будет вынужден сделать эти разрывы, исходя из формального смысла своих алгоритмов. При этом, скорее всего, на печати вы получите много страниц, разорванных в самых неожиданных и неудачных с точки зрения человека местах, а на экране — много сообщений вроде такого:

```
Underfull \vbox (badness 10000) has occurred
while \output is active
```

Если вы регулярно сталкиваетесь с такими неприятностями, имеет смысл заново продумать принципы организации вашего текста. Избавиться от растянутых по вертикали страниц можно, если дать в преамбуле документа команду `\raggedbottom`, разрешающую делать страницы неодинаковой высоты (некоторые классы документов дают эту команду автоматически, но если вы ее продублируете, ничего плохого не случится). Действие, противоположное `\raggedbottom`, вызывается командой `\flushbottom`.

Наконец, если трудности возникают оттого, что вы часто оставляете в тексте место для рисунка командой `\vspace*`, то вам стоит воспользоваться «плавающими» иллюстрациями (см. разд. IV.8).

10. Линейки

10.1. Линейки в простейшем виде

Один из часто встречающихся элементов полиграфического оформления — так называемые «линейки». Например, в книге, которую вы читаете, колонтитулы отделены линейкой от основной части страницы. В TeX'е линейкой (rule по-английски) называется любой черный прямоугольник. Для создания линеек в LaTeX'е используется команда `\rule`. У этой команды два обязательных аргумента: первый задает ширину прямоугольника-линейки, второй — высоту (оба этих размера должны быть заданы в используемых TeX'ом единицах измерения — см. с. 26). Линейка, созданная командой `\rule`, рассматривается TeX'ом так же, как буква.

Если необходимо, чтобы созданный командой `\rule` прямоугольник был сдвинут по вертикали относительно уровня строки, надо воспользоваться командой `\rule` с необязательным аргументом. Этот аргумент — расстояние, на которое надо сдвинуть линейку по вертикали, — ставится *перед* обязательными; если расстояние положительное, то сдвиг идет вверх, если отрицательное, то вниз. Пример:

В этом месте, прямо посреди абзаца, будет линейка ▌, а после нее продолжится обычный текст. Сравните также ■ и ■

```
В этом месте, прямо посреди
абзаца, будет линейка
\rule{.5em}{15pt}, а
после нее продолжится
обычный текст. Сравните
также \rule{5pt}{5pt}
и \rule[-3pt]{5pt}{5pt}
```

10.2. TeX'овские команды для генерации линеек

Рассмотренная нами LaTeX'овская команда `\rule` обладает рядом недостатков. Например, то обстоятельство, что создаваемые с ее помощью линейки воспринимаются TeX'ом как буквы, усложняют печать линейки, простирающейся во всю ширину страницы. Если между абзацами, т. е. в «вертикальном режиме», сказать `\rule{10cm}{1pt}`, то линейка начнется не с левого края текста, а после абзацного отступа: TeX решит, что с этой «буквы» начинается новый абзац. Кроме того, при печати линеек с помощью команды `\rule` необходимо заранее знать их длину и ширину, что не всегда удобно (например, если линейка должна идти во всю ширину текста, то надо точно знать, чему эта ширина равна, или, по крайней мере, как она обозначается в TeX'е). Избавиться от этого неудобства можно с помощью TeX'овских команд `\hrule` и `\vrule`. Команда `\hrule` употребляется в «вертикальном режиме» (между абзацами). Она создает линейку высотой `0.4pt` и шириной, равной ширине колонки текста. Команда `\vrule` употребляется в «горизонтальном режиме» (внутри абзацев). Она создает линейку шириной `0.4pt`, простирающуюся по высоте до максимальной высоты букв в содержащей ее строке (если в строке присутствуют буквы наподобие «у», опускающиеся ниже уровня строки, то и линейка будет опускаться ниже уровня строки). Пример:

Весь этот текст будет заключен между двумя линейками. Внутри абзаца тоже будет | линейка.

Если буквы в строках выше, то и линейка | будет больше.

```
\hrule\smallskip
Весь этот текст будет заключен
между двумя линейками.
Внутри абзаца тоже будет
\vrule{} линейка.

\Large Если буквы в строках
выше, то и линейка \vrule{}
будет больше. \smallskip\hrule
```

Если вас не устраивает, что генерируемая командой `\hrule` линейка имеет высоту `0.4pt`, то требуемую вам высоту можно указать в явном виде. Например, для задания линейки шириной во всю колонку и высотой 2 пункта надо написать (как водится, между абзацами) так:

```
\hrule height 2pt
```

Отсутствие символа \ перед `height` не является опечаткой (`height` — не команда, а одно из так называемых «ключевых слов» TeX'а, наподобие уже встретившихся нам в разд. 9.4 слов `plus` и `minus`). Для явного задания ширины линейки, генерируемой командой `\vrule`, используется ключевое слово `width`:

```
\vrule width 2mm
```

В принципе можно указывать при команде `\hrule` не только высоту, но и ширину, а при команде `\vrule` — не только ширину, но и высоту, но в таком случае обычно проще воспользоваться LaTeX'овской командой `\rule`.

Если после команды `\hrule` или `\vrule` в тексте идет слово, совпадающее с одним из используемых этими командами ключевых слов (то бишь `height`, `width` или `depth`, о котором у нас речи не было), то это слово будет воспринято TeX'ом как ключевое, что приведет к сообщению об ошибке. В русском тексте вероятность такого стечения обстоятельств исчезающе мала, но если вы хотите, чтоб неприятностей не было с гарантией, то после чего-нибудь вроде `\hrule height 2mm` пропустите строку (между абзацами это ничего не испортит), а после команды наподобие `\vrule width 2mm` поставьте еще команду `\relax`, означающую «ничего не делать».

10.3. Невидимые линейки

Высота и/или ширина линейки может быть и нулевой. Линейки нулевой высоты или ширины не печатаются, но тем не менее могут оказать влияние на вид текста. Например, линейка нулевой ширины и ненулевой высоты занимает место по вертикали; если ее высота больше высоты букв в строке, то высота строки, содержащей эту невидимую линейку, увеличится:

Для того, чтобы раздвинуть две строки, можно в одну из них поместить невидимую линейку нулевой ширины и ненулевой высоты.

```
Для того, чтобы раздвинуть две
строки, можно в одну из них
поместить невидимую
линейку\rule{0pt}{5mm} нулевой
ширины и ненулевой высоты.
```

Один частный случай линейки нулевой ширины настолько важен, что в TeX'е и LaTeX'е для такой линейки предусмотрена специальная команда `\strut`. Невидимая линейка, создаваемая этой командой, имеет нулевую ширину; высота же ее установлена автором LaTeX'а с таким расчетом, чтобы она была чуть выше максимальной высоты букв текущего шрифта и опускалась ниже уровня строки настолько, насколько могут опускаться буквы текущего шрифта. Например, в прямом светлом шрифте кегля 11 команда `\strut` создает линейку ширины 0, поднимающуюся над уровнем строки на 9.52 pt и опускающуюся ниже уровня строки на 4.08 pt.

Линейки нулевой ширины и ненулевой высоты действуют подобно команде `\vspace*`. Смысл невидимых линеек в том, что они позволяют создать вертикальные или горизонтальные пробелы в таких ситуациях, когда `\vspace` или `\hspace` не помогают. Вот пример ситуации, когда возникает нужда в невидимых линейках. Пусть в нашем тексте мы подчеркнули три слова подряд. Выглядит это не очень удачно: в словах с буквами вроде р, опускающимися ниже строки, линейки, подчеркивающие слово, также опускаются ниже строки, а хотелось бы, чтобы все эти линейки были на одном уровне. Выход из положения такой: добавить ко всем словам по невидимой букве, которая не занимает места по горизонтали, а по вертикали опускается на максимально возможное в текущем шрифте расстояние. В качестве такой буквы как раз и возьмем невидимую линейку, генерируемую командой `\strut`:

три слова подряд

```
\underline{три\strut}
\underline{\strut слова}
\underline{подряд\strut}
```

Как видите, `\strut` можно ставить хоть после слова, хоть перед ним (и даже посередине, если вы не запутаетесь с пробелами). Назначение этой команды в данном случае сводится к тому, чтобы не позволить линейке, подчеркивающей слово, подойти к этому слову слишком близко. Кстати, в переводе с английского слово strut означает «распорка».

В гл. II рассказывалось про команду `\mathstrut`, выполняющую аналогичные функции в математических формулах.

Другие примеры использования невидимых линеек читатель найдет в главе VI, посвященной набору таблиц; в главе VIII, посвященной «блокам», мы также встретимся с линейками.

Глава IV

Оформление текста в целом

В этой главе мы рассмотрим такие вопросы, как общий стиль оформления документа, разбиение текста на разделы, титульный лист, оглавление и пр. Система LaTeX освобождает вас от многих забот об оформлении документа, но при этом и навязывает такие черты оформления, которые могут вас по тем или иным причинам не устраивать. От этого «диктата» можно отчасти отойти, если модифицировать стандартные классы, создав свой стилевой пакет (в последней главе мы расскажем, как это делать). В принципе можно создать и свой собственный класс, весьма далекий от стандартных, но для этого требуются более глубокие познания в TeX'е, чем дает эта книга. Начнем же мы с того, что систематически рассмотрим вопрос о стандартных классах.

1. Классы, пакеты и классовые опции

Команда `\documentclass`, с которой начинается любой LaTeX'овский файл, имеет один обязательный аргумент — название основного класса — и один необязательный, размещающийся перед обязательным, — список, через запятую, «классовых опций» (см. с. 20).

Прежде чем говорить о классах документов, скажем несколько слов о команде, которая очень часто идет сразу после `\documentclass`, а именно, команде `\usepackage`. После `\documentclass` может идти одна или несколько команд `\usepackage`; аргумент этой команды — это список, через запятую, стилевых пакетов, подключаемых к нашему документу. В первой главе мы умолчали о том, что некоторые стилевые пакеты допускают задание своих личных стилевых опций (каких именно — зависит от пакета). Список стилевых опций пакета задается в необязательном аргументе команды `\usepackage` (через запятую, если опций несколько). Необязательный аргумент команды `\usepackage` ставится перед обязательным.

Например, если включить в преамбулу строку

```
\usepackage{amsmath}
```

то вам откроются дополнительные возможности набора математических формул, о которых мы много рассказывали во второй главе; если же написать

```
\usepackage[noamsfonts]{amsmath}
```

то у вас будут все эти возможности, кроме использования готического и ажурного шрифтов.

Некоторые стилевые пакеты в своей работе, кроме того, учитывают «классовые опции» (это то, что указывается в необязательном аргументе команды `\documentclass` — см. ниже).

Теперь вернемся к классам документов. В первой главе мы уже перечислили пять классов, предоставляемых LaTeX'ом: `article`, `report`, `book`, `proc` и `letter`. Сейчас мы рассмотрим подробнее четыре из них. Класс `letter` рассматривается ниже, в разд. 2. (Американское математическое общество распространяет также, вместе со своими стилевыми пакетами, классы документов `amsart`, `amsproc` и `amsbook`; мы рассмотрим их в приложении Г.)

Класс `article` удобно применять для статей, класс `report` — для более крупных статей, разбитых на главы, или небольших книг, класс `book` — для книг. В табл. IV.1 перечислены некоторые черты оформления, присущие стандартным классам. В ней знак «$+$» означает «всегда присутствует», знак «$-$» означает «всегда отсутствует», знак «$\mp$» означает «по умолчанию отсутствует, но будет присутствовать, если задать классовую опцию или специальную команду», знак «$\pm$» означает «по умолчанию присутствует, но можно отменить с помощью классовой опции или специальной команды». Мы не стремились охватить в этой таблице все детали различий между стандартными классами (например, колонтитулы в разных классах оформляются по-разному).

Опишем теперь классовые опции. Напомним (см. с. 20), что список классовых опций через запятую ставится в квадратных скобках перед основным аргументом команды `\documentclass`. Самые часто употребляемые классовые опции — это `11pt` и `12pt`. Они означают, что основной текст документа будет набран шрифтом кегля 11 или 12 соответственно. Если этих опций не указывать, то будет шрифт кегля 10.

Можно указать классовую опцию, задающую формат используемой бумаги, после чего TeX рассчитает размеры текста и полей так, чтобы

Таблица IV.1.

	article	report	book	proc
Автоматически нумерующиеся разделы	+	+	+	+
Разбиение на главы	−	+	+	−
«Двусторонняя»* печать	∓	∓	±	−
Титульный лист	∓	±	±	−
Колонтитулы	∓	±	±	∓
Одинаковая высота всех страниц	∓	∓	±	−
Набор в две колонки	∓	∓	∓	+
Набор в одну колонку	±	±	±	−

*с разными полями для четных и нечетных страниц.

они максимально соответствовали этому формату. Эти опции таковы (единица измерения указана в скобках):

`a4paper` 210×297 (миллиметры) — самый ходовой в нашей стране;

`a5paper` 148×210 (миллиметры);

`b5paper` 176×250 (миллиметры);

`legalpaper` 8.5×14 (дюймы);

`executivepaper` 7.25×10.5 (дюймы).

Если ни одна из этих опций не указана, LaTeX считает, что размер бумаги равен (в дюймах) 8.5×11 (этот формат бумаги иногда называют «letter»).

Если предполагается расположить текст так, чтобы он шел параллельно широкому, а не узкому краю бумаги, то можно указать опцию `landscape`: в этом случае TeX будет вычислять размеры текста и полей, считая, что ширина и высота листа бумаги поменялись ролями. Подчеркнем, что задание опции `landscape` само по себе текст на 90° не повернет: он будет сверстан TeX'ом исходя из соответствующих размеров, но дальше необходимо иметь принтер и/или `dvi`-драйвер, способные обеспечить печать текста в такой ориентации. По умолчанию же считается, что строки параллельны узкому краю листа.

Опция `twoside` задает печать с разными полями на нечетных и четных страницах (как в книгах), а опция `oneside` — печать с одинаковыми полями на всех страницах. В классе `book` по умолчанию установлена опция twoside, в классах article и report — oneside, в классе proc поля на четных и нечетных страницах всегда одинаковые.

Для классов article, report и book можно указать классовую опцию twocolumn (см. разд. III.9.6). Она означает, что набор текста будет производиться в две колонки. Так как абзацы при этом будут получаться довольно узкие, разумно при пользовании этой опцией заодно увеличить параметр \tolerance (см. с. 125), иначе будет получаться много строк, выбивающихся за колонку. В разд. III.9.6 объяснялось, как сделать так, чтобы две колонки отделялись линейкой. В классе proc набор всегда производится в две колонки, и опции twocolumn и onecolumn (есть и такая) на него не действуют.

Опция draft пригодна для любого класса. Если она включена, то каждая выбивающаяся на поля строка (т. е. строка, о которой выдается сообщение «Overfull \hbox» — см. с. 116) помечается на полях «марашкой» ▮. Это удобно при подготовке корректур (английское слово draft как раз и означает «набросок»).

Режим, при котором разные страницы могут иметь разную высоту, задается, как мы помним, командой \raggedbottom (см. разд. III.9.7). По умолчанию этот режим устанавливается классами article и report, если только в качестве классовой опции не указана «двусторонняя» печать (классовая опция twoside), а также классом proc. Во всех остальных случаях LaTeX будет, по умолчанию, делать все страницы одинаковой высоты.

У классов report и book имеются опции openright и openany. Если указана опция openright, то каждая глава начинается обязательно с нечетной страницы (если необходимо, то ради этого печатается дополнительная пустая страница; на развороте нечетная страница будет правой). Если указана опция openany, то новая глава может начинаться как с четной, так и с нечетной страницы, и лишних пустых страниц ради начала главы LaTeX не делает. По умолчанию в классе report LaTeX действует так, как если бы было установлено openany, а в классе book — как если бы было openright.

Скажем кое-что про класс документов proc. В этом классе текст печатается в две колонки, с уменьшенными полями. Опции a5paper, b5paper, onecolumn и titlepage в классе proc использовать нельзя. Нельзя также пользоваться маргиналиями (разд. 10).

При пользовании классом proc внизу каждой страницы будет напечатано слово Page («страница») и номер страницы. Если вы хотите, чтобы вместо Page печаталось что-то другое, скажем, «с.», то нужно в преамбуле переопределить команду \pagename, то есть написать:

```
\renewcommand{\pagename}{c.}
```

(см. с. 169 по поводу переопределения «стандартных надписей», делаемых LaTeX'ом по-английски; `\renewcommand` обсуждается в гл. VII).

Следующие две классовые опции, применимые к любому из основных классов, влияют на оформление выключных математических формул; если вы пропустили при чтении соответствующие места в гл. II, то пропустите и это место. Опция `fleqn` означает, что выключные формулы, заданные с помощью окружений `equation`, `eqnarray` и `displaymath`, а также пары команд `\[` и `\]`, будут напечатаны не в центре строки, а в ее левой части. Опция `leqno` означает, что номера формул, генерируемые окружениями `equation` и `eqnarray`, будут печататься не справа, а слева.

Титульному листу и оформлению частей документа будут посвящены отдельные разделы; чтобы завершить наш обзор различных стандартных вариантов стиля, нам осталось обсудить колонтитулы и номера страниц, чему будет посвящен разд. IV.3.

2. Деловые письма

```
\documentclass[11pt]{letter}
\address{I.\,Ivanov,\\
  IUM, Moscow, Russia}
\signature{I.\,Ivanov}
\date{December 31, 2002}
\begin{document}
\begin{letter}{Prof.\ Jonas Jonas Huckabuck,\\
  University of Rootabaga,\\
  Cream Puffs, 80116, RO, USA}
\opening{Dear Prof.\ Huckabuck,}
Enclosed please find two copies of my paper.
\closing{Sincerely yours,}
\end{letter}
\end{document}
```

Рис. IV.1. Пример делового письма

Существует класс документов `letter`, специально предназначенный для оформления деловых писем в соответствии с принятыми в США стандартами. Пользоваться им надо так. Во-первых, в преамбуле документа надо задать адрес отправителя в аргументе команды `\address` (можно применять `\\` для ручного разбиения адреса на

I. Ivanov,
IUM, Moscow, Russia

December 31, 2002

Prof. Jonas Jonas Huckabuck,
University of Rootabaga,
Cream Puffs, 80116, RO, USA

Dear Prof. Huckabuck,

Enclosed please find two copies of my paper.

Sincerely yours,

I. Ivanov

Рис. IV.2. Письмо с рис. IV.1 в напечатанном виде

строки) и подпись отправителя в аргументе команды `\signature`. Можно также задать дату в аргументе команды `\date`. Если этого не сделать, то автоматически будет проставлена дата трансляции LaTeX'овского файла (по-английски).

Между `\begin{document}` и `\end{document}` идет собственно текст письма или писем (в одном LaTeX'овском файле можно поместить несколько писем от одного отправителя). Каждое письмо оформляется как окружение `letter`, принимающее один обязательный аргумент — адрес того, кому предназначено письмо (опять-таки адрес можно разбивать на строки командами `\\`). Внутри окружения `letter` располагается собственно текст письма. Начинаться он должен командой `\opening`, в аргументе которой записывается вступительное обращение, а завершаться — командой `\closing`, в аргументе которой записывается заключительная фраза перед подписью («Sincerely yours», например). На рис. IV.1 приведен пример оформленного по всем правилам LaTeX'овского файла с письмом, а на рис. IV.2 — то, как будет выглядеть это письмо на печати. Вертикальный отступ на печати между текстами, заданными в командах `\closing` и `\signature` (в нашем случае — между «Sincerely yours» и «I. Ivanov») предназначен для того, чтобы поставить в этом месте подпись.

Если вы хотите написать в письме что-то после вашей подписи, следует после команды `\closing` дать команду `\ps` («постскриптум»), и уже после нее — сам текст (команда `\ps` необходима, но никакого текста, даже просто P.S., она не генерирует).

В преамбуле можно дать команду `\makelabels`. В этом случае на отдельной странице будут напечатаны адреса для всех писем (окружений `letter`), входящих в файл.

Все описанные выше классовые опции применимы и к документам класса `letter`.

После команды `\closing` можно дать команду `\cc`, в аргументе которой указывается, кому вы собираетесь отослать копии письма, а также команду `\encl`, в аргументе которой указывается список вложений. Американский стандарт на оформление таких вещей расходится с отечественным. Если, тем не менее, вы решитесь воспользоваться этими средствами в письме на русском языке, следует изменить значение команд `\ccname` и/или `\enclname` с помощью `\renewcommand` (см. образцы в разд. 5.3).

3. Стиль оформления страницы

Для задания стиля оформления страницы в LaTeX'е предусмотрена команда `\pagestyle`. Эта команда имеет один обязательный аргумент — слово, обозначающее этот стиль. При пользовании стандартными классами документов это слово должно быть одним из следующих:

`empty`	нет ни колонтитулов, ни номеров страниц;
`plain`	номера страниц ставятся внизу в середине строки, колонтитулов нет;
`headings`	присутствуют колонтитулы (включающие в себя и номера страниц);
`myheadings`	присутствуют колонтитулы, оформленные так же, как в предыдущем случае; отличие в том, что текст, печатающийся в колонтитулах (в стандартном случае это номера и названия разделов документа), не порождаются LaTeX'ом автоматически, а задается пользователем в явном виде.

Если основной стиль — `article`, то по умолчанию страницы оформляются стилем `plain`, в двух других основных стилях — стилем `headings`. «Стиль» `myheadings` мы рассмотрим в разд. IX.6.

Наряду с командой `\pagestyle`, задающей стиль оформления всех страниц, есть и команда `\thispagestyle`, задающая стиль оформления одной отдельно взятой страницы. Она принимает такой же аргумент, как и \pagestyle, но указываемое этим аргументом оформление относится только к той странице, на которую попал текст, окружающий эту команду. Заранее предугадать, на какую страницу попадет данный фрагмент текста, обычно невозможно. Поэтому, если хотите от этой команды предсказуемых результатов, употребляйте ее непосредственно после \newpage или \clearpage.

Можно при желании сделать так, чтобы страницы нумеровались не арабскими цифрами, что делается по умолчанию, а римскими цифрами или буквами в алфавитном порядке. Для этого предназначена команда \pagenumbering. Она имеет один обязательный аргумент, который может быть одним из следующих:

arabic	арабские цифры (1, 2, 3,...)
roman	римские цифры (i, ii, iii,...)
Roman	римские цифры (I, II, III,...)
alph	строчные буквы (a, b, c,...)
Alph	прописные буквы (A, B, C,...)

Команда \pagenumbering не только меняет вид, в котором на печати представляются номера страниц, но и начинает счет страниц заново (это удобно, например, в тех случаях, когда страницы предисловия надо нумеровать римскими цифрами, а страницы основного текста заново нумеровать арабскими). Поэтому разумно давать эту команду сразу же после \newpage или \clearpage.

На этом мы прерываем наше обсуждение того, как изменять стандартное оформление страницы. На самом деле можно изменить гораздо больше, но речь об этом пойдет в гл. IX.

4. Поля, размер страницы и прочее

Класс документа предопределяет значения таких параметров, как ширина и высота страницы, размеры полей и пр. (с учетом опции, указывающей формат бумаги). В настоящем разделе рассказано, как изменить эти значения, если они вас не устраивают.

Размеры текста на странице, полей и пр. задаются параметрами со значением длины (см. разд. I.2.9 по поводу TEX'овских параметров). Менять эти параметры следует в преамбуле документа.

4.1. Ширина

Ширина текста на странице задается параметром \textwidth; если набор осуществляется в две колонки, то \textwidth включает в себя ширину обеих колонок и пробел между ними. Если нужно, чтобы ширина текста на странице была 7 сантиметров, напишите в преамбуле так:

```
\textwidth=7cm
```

При изменении ширины текста часто приходится менять и поля. Для этого предусмотрен параметр, регулирующий размер левого поля (коль скоро левое поле и `\textwidth` заданы, правое поле определяется автоматически). Способ задания левого поля зависит от того, является ли набор в данном стиле «двусторонним» или нет. На с. 157 объяснялось, что при двустороннем наборе на страницах с четными и нечетными номерами оставляются разные поля. В классах документов `article` и `report` набор по умолчанию односторонний, но он будет двусторонним, если указать стилевую опцию `twoside`. В классе `book` набор по умолчанию двусторонний, но можно сделать его односторонним, если указать стилевую опцию `oneside`.

При одностороннем наборе величина левого поля задается параметром `\oddsidemargin`. При этом поле отсчитывается не от самого края листа: предварительно делается отступ в один дюйм. Таким образом, если вы скажете в преамбуле

```
\oddsidemargin=0pt
```

то текст будет начинаться на расстоянии один дюйм от края, а если будет сказано

```
\oddsidemargin=5mm
```

то отступ от края бумаги составит 30.4 мм (вспомним, что один дюйм равен 2.54 см). Если присвоить параметру `\oddsidemargin` отрицательное значение, то расстояние от края листа до начала текста будет, соответственно, меньше дюйма. Нелишне также напомнить, что когда вы присваиваете параметру со значением длины нулевое значение, то все равно должна быть указана какая-то единица длины (как у нас в примере); запись наподобие

```
\oddsidemargin=0
```

является ошибочной.

Все сказанное относилось к одностороннему набору. При двустороннем наборе параметр `\oddsidemargin` также используется, но смысл его несколько иной: на сей раз он задает размеры левого поля только для страниц с нечетными номерами. Что же касается страниц с четными номерами, то размеры левого поля для них задаются параметром `\evensidemargin`.

При наборе текста в две колонки используются еще два параметра. Во-первых, параметр `\columnsep` задает расстояние между колонками; во-вторых, колонки можно при желании разделить не только пробелом, но и вертикальной линейкой. Ширина этой линейки задается параметром `\columnseprule`. В стандартных стилях значение этого последнего параметра установлено равным нулю, так что линейка между колонками не печатается; чтобы линейка была, надо в преамбуле присвоить параметру `\columnseprule` значение, отличное от нуля (в этом случае ширина разделяющей колонки линейки включается в `\columnsep`). Хорошее ненулевое значение параметра `\columnseprule` — `0.4pt`.

4.2. Высота

Размер верхнего поля задается параметром `\topmargin`; как и в случае с левым полем, это — расстояние не непосредственно от края листа, а от линии, параллельной краю и отстоящей от него на один дюйм. При этом надо сознавать не только от чего, но и до чего отсчитывается это расстояние: именно, `\topmargin` — это расстояние до колонтитула. Если же колонтитул на странице отсутствует (например, потому, что он не предусмотрен стилем), то вверху страницы дополнительно будет пустое пространство, размер которого равен месту, отводимому на колонтитул (параметр `\headheight`, как мы узнаем из гл. IX) плюс отступ между колонтитулом и основным текстом (параметр `\headsep`). Высота текста задается параметром `\textheight`. При исчислении этого размера не учитываются ни номера страниц, ни колонтитулы, так что, если они предусмотрены классом, полная высота текста на странице будет больше, чем `\textheight`.

Высоту страницы также можно изменять, присваивая в преамбуле параметру `\textheight` новое значение, но если класс предусматривает, что все страницы должны иметь одинаковую высоту (см. с. 157 и ниже по поводу того, когда именно так бывает), то высоту текста нельзя устанавливать совсем уж произвольно: необходимо согласовать ее значение с параметрами `\topskip` и `\baselineskip`. Не вдаваясь в подробности, скажем, что первый из этих параметров определяет расстояние от низа первой строки[1] до «верхнего обреза» основного текста страницы, в то время как параметр `\baselineskip` определяет расстояние между строками и зависит от используемого шрифта (будем надеяться, что вы не станете менять его значение, не изучив предварительно книгу [2]). Так или иначе, значение `\textheight` следует

[1] Точнее говоря, от ее базисной линии: см. гл. VIII.

устанавливать таким образом, чтобы отношение

$$\frac{\texttt{\textbackslash textheight} - \texttt{\textbackslash topskip}}{\texttt{\textbackslash baselineskip}}$$

было целым числом. В LaTeX'овском стандарте `\topskip` всегда равен 10 пунктам. Что же до `\baselineskip`, то он равен 12 пунктам, если основной шрифт кегля 10, 13.6 пункта, если основной шрифт кегля 11, и 15 пунктам в кегле 12.

Можно и не знать точно этих размеров, но вычислить нужную величину `\textheight` средствами самого TeX'а: именно, если вы, скажем, хотите, чтобы на странице помещалось 40 строк, то напишите в преамбуле (после команды, изменяющей интерлиньяж, если вы таковой пользовались) следующее:

```
\setlength{\textheight}{40\baselineskip}
\setlength{\textheight}{\baselinestretch\textheight}
\addtolength{\textheight}{\topskip}
```

(см. разд. VII.4 по поводу `\setlength` и `\addtolength`).

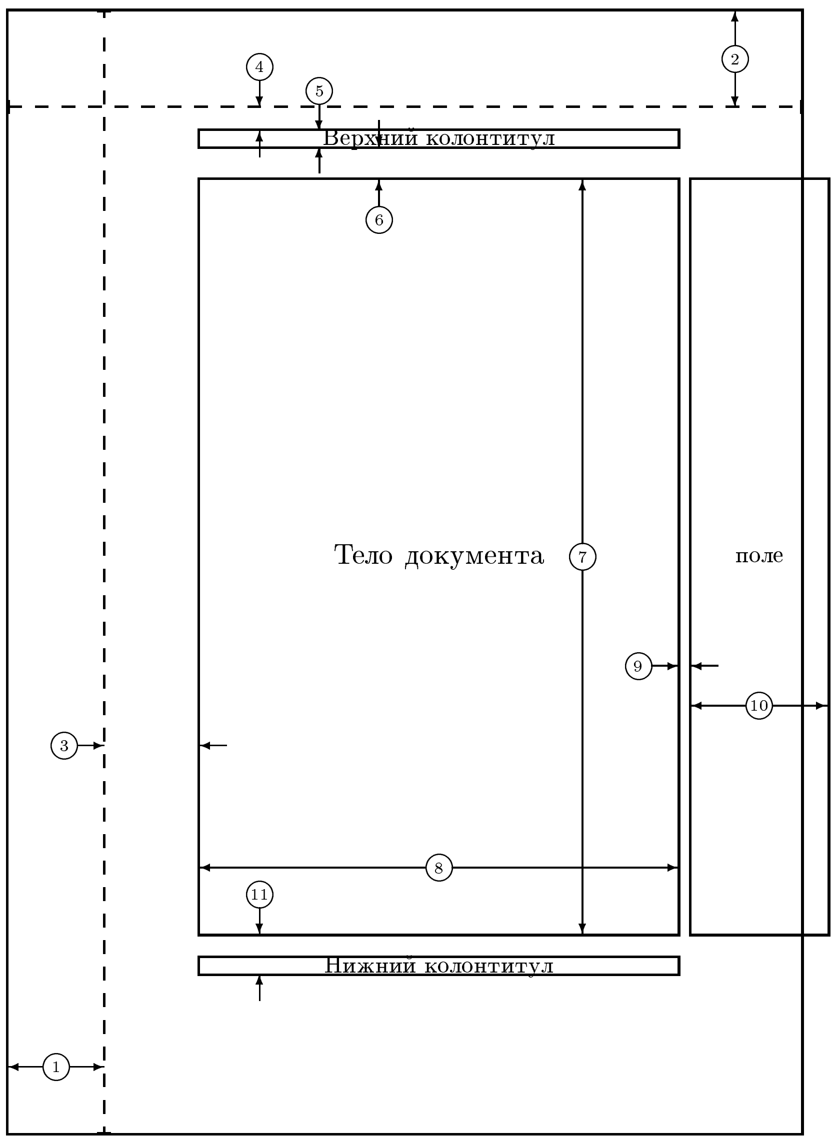
4.3. Сдвиг страницы как целого

Иногда при печати вы можете обнаружить, что реальные расстояния от текста до края листа не такие, как предписано параметрами наподобие `\topmargin`. Дело в том, что у принтера, которым вы пользуетесь, могут быть свои представления о том, где находится край листа бумаги. Чтоб эти представления соответствовали реальности, понадобится настройка принтера и/или `dvi`-драйвера, используемого вами для распечатки. Если вы не хотите этим заниматься, можно просто изменить расположение всей страницы в целом на печатном листе. Для этого следует установить (в преамбуле) значения двух TeX'овских параметров: `\hoffset` и `\voffset`. Например, если в преамбуле написано

```
\hoffset=-5mm
\voffset=4.2mm
```

то вся страница в целом (со всеми колонтитулами, номерами страниц и пр.) будет сдвинута при печати на 5 мм влево и на 4,2 мм вниз.

Если вы запутались во всех этих параметрах, посмотрите на изготовленную с помощью пакета `layout` картинку на с. 166, на которой изображена геометрия страницы LaTeX'а (с теми значениями параметров, которые установлены в нашем тексте). О некоторых из указанных на ней параметров мы пока не упоминали; речь о них пойдет далее в этой главе.



1	1 дюйм + \hoffset	2	1 дюйм + \voffset
3	\oddsidemargin = 72pt	4	\topmargin = 18pt
5	\headheight = 12pt	6	\headsep = 25pt
7	\textheight = 568pt	8	\textwidth = 360pt
9	\marginparsep = 10pt	10	\marginparwidth = 103pt
11	\footskip = 30pt		\marginparpush = 5pt (не показано)
	\hoffset = 0pt		\voffset = 0pt
	\paperwidth = 597pt		\paperheight = 845pt

5. Рубрикация документа

Работая с LaTeX'ом, разумно делать заголовки и нумерацию разделов документа не вручную, а с помощью специальных команд. Сначала разберем, как ими пользоваться, на примере команды `\section`.

5.1. Команда \section

Пусть вам нужно начать раздел документа, озаглавленный «Кое-что о слонах». Для этого в исходном тексте можно написать так:

```
\section{Кое-что о слонах}
```

Команда `\section` принимает один обязательный аргумент — название раздела (это же название пойдет в колонтитулы, если таковые предусмотрены классом, и в оглавление, если вы дадите команду «создать оглавление» — см. ниже с. 173). Промежутки между разделами, их нумерация, те же колонтитулы — все это делается автоматически.

Кроме обязательного аргумента, у команды `\section` предусмотрен и необязательный. Необязательный аргумент идет перед обязательным; в нем записывается вариант заголовка, предназначенный для оглавления и колонтитулов (если класс предусматривает, что заголовок войдет в колонтитул). Обычно этот вариант — просто сокращенный заголовок. Пример:

```
\section[О слонах]{Кое-что о слонах}
```

Необходимость в сокращенном варианте заголовка возникает, когда оказывается, что заголовок по длине не помещается в колонтитул. Это, конечно, будет видно при просмотре; кроме того, при трансляции в этом случае выдается такое сообщение:

```
Overfull \hbox has occurred while \output was active.
```

Раздел можно пометить командой `\label` (см. с. 27). После этого команда `\ref` будет выдавать номер этого раздела. Пример:

3.2 Кое-что о слонах

В этом разделе нашей книги речь пойдет в основном о слонах. Слоны (см. определение в разд. 3.2) — большие и сильные животные.

```
\section{Кое-что о слонах}
\label{elephants}
В этом разделе нашей книги
речь пойдет в основном о
слонах. Слоны
(см. определение в
разд.~\ref{elephants})~---
большие и сильные животные.
```

Как обычно с командами автоматической генерации ссылок, при первом запуске LaTeX'а будет выдано сообщение о том, что метка неизвестна, а в дальнейшем, если номер помеченного раздела изменится, LaTeX выдаст предупреждение о том, что надо запустить его еще раз.

Иногда хочется, чтобы сокращенный вариант заголовка попал только в колонтитул, а в оглавлении был его полный вариант. Как этого добиться, рассказано в разд. IX.6 (см. с. 325).

У команды `\section` есть вариант «со звездочкой» (см. с. 25). Команда `\section*` начинает новый раздел, не нумеруя его; на оглавлении и колонтитулах наличие раздела, вводимого этой командой, никак не отразится. У команды `\section*` предусмотрен только обязательный аргумент.

5.2. Какие бывают разделы документа

Теперь перечислим все команды для задания разделов документа, предоставляемые стандартными классами LaTeX'а. Большинство из них работает совершенно аналогично команде `\section`; все отличия мы сейчас перечислим.

Для оформления разделов существуют такие команды:

```
\part \chapter \section \subsection
\subsubsection \paragraph \subparagraph
```

В этом перечне каждая последующая команда обозначает более мелкий подраздел, чем предыдущая. Следует иметь ввиду, что команда `\chapter` («глава») в классах `proc` и `article` не определена (благодаря этому обстоятельству статью легко переделать в главу книги), остальные команды определены в четырех основных классах.

Стандартные классы обеспечивают нумерацию разделов, при которой более мелкий раздел «подчинен» более крупному: когда, например, начинается новый раздел `\section`, нумерация разделов `\subsection` и более мелких начинается заново. Исключением из этого правила является команда `\part` («часть»): если часть 2 кончается главой 5, то первая из глав части 3 будет иметь номер 6, а не 1. При модификации стандартных классов можно менять как принцип нумерации разделов, так и вид этих «номеров» на печати (например, если мы захотим, чтобы разделы обозначались последовательными буквами алфавита).

Все то, что мы говорили про необязательный аргумент и вариант «со звездочкой» у команды `\section`, применимо и к командам, перечисленным в этом разделе. «Слишком мелкие» разделы, согласно стандартным классам, не отражаются ни в оглавлении, ни в колонтитулах и не нумеруются, но, если вы употребите задающие их команды с необязательным аргументом или со звездочкой, ошибкой это не будет.

В разделах, создаваемых описанными выше командами, первый абзац набирается без абзацного отступа (за исключением самого мелкого раздела `\subparagraph`), причем LaTeX устроен таким образом, что создать этот отступ вам так просто не удастся. Если вы хотите, чтобы отступ в первом абзаце все-таки присутствовал, обратитесь к гл. IX, посвященной модификации стандартных классов.

5.3. Изменение стандартных заголовков

Возможно, вы уже обратили внимание, что главы, создаваемые LaTeX'ом с помощью команды `\chapter`, так и называются «Chapter», а не «Глава». Это — один из нескольких случаев, когда классы LaTeX'а используют для различных стандартных надписей английские слова. В некоторых русификациях LaTeX'а предусмотрены стилевые пакеты, переименовывающие эти названия в русские.

Если такой русификации у вас нет, то придется решить эту проблему своими силами. Для этого в преамбулу документа надо внести следующую команду:

```
\renewcommand{\chaptername}{Глава}
```

Делать это надо, конечно, только в том случае, если команда `\chapter` предусмотрена в используемом вами классе. О команде `\renewcommand`, используемой для переопределения значений уже существующих команд, рассказывается в гл. VII.

Приведем список остальных надписей, делаемых LaTeX'ом по-английски, вместе с их русскими переводами и командами, которые надо переопределить с помощью `\renewcommand` для того, чтоб на печати появлялись именно эти переводы. В нескольких ближайших разделах мы объясним, как именно получить на печати упоминаемые в этой таблице список таблиц, список иллюстраций и т. п.

`\chaptername`	Chapter	Глава
`\contentsname`	Contents	Оглавление (или Содержание)
`\listfigurename`	List of Figures	Список рисунков
`\listtablename`	List of Tables	Список таблиц
`\abstractname`	Abstract	Аннотация
`\refname`	References	Литература
`\bibname`	Bibliography	Литература
`\indexname`	Index	Предметный указатель
`\figurename`	Figure	Рис.
`\tablename`	Table	Таблица
`\partname`	Part	Часть
`\appendixname`	Appendix	Приложение

Пояснений эта таблица не требует, за одним исключением: если класс документа есть `article` или `proc`, то для получения русского названия списка литературы надо переименовать команду `\refname`, а если `report` или `book` —

то команду `\bibname`. Пожалуйста, не перепутайте эти два случая, иначе LaTeX зафиксирует ошибку и выполнять вашу команду откажется.

5.4. До и после основного текста

В классах `article`, `report` и `proc` предусмотрена возможность оформить аннотацию ко всему документу. Это делается с помощью окружения `abstract`. До начала основного текста следует поместить текст аннотации между `\begin{abstract}` и `\end{abstract}`. Этот текст будет автоматически озаглавлен «Abstract», если не подключать русифицирующий стилевой пакет и если вы сами не переопределили команду `\abstractname` (см. предыдущий пункт).

Команда `\appendix` означает, что с этого места начинается приложение к документу. Сама она никакого текста не производит, а делает вот что:

- начинает заново нумерацию разделов документа;
- «самые крупные» разделы документа (`\section` в классах `article` и `proc`, `\chapter` в двух других основных классах) начинают нумероваться не цифрами, а прописными латинскими буквами;
- если определена команда `\chapter`, то главы будут с этого момента называться не «Chapter», а так, как определено в команде `\appendixname` (см. предыдущий пункт).

Если класс вашего документа — `book`, то можете воспользоваться услугами команд `\frontmatter`, `\mainmatter` и `\backmatter`. Командой `\frontmatter` открывается «вводная часть» книги: после этой команды страницы начинают нумероваться римскими цифрами (как если бы мы сказали `\pagenumbering{roman}`). В той части текста, на которую распространяется действие команды `\frontmatter`, главы не нумеруются. Переход к основной части книги задается командой `\mainmatter`: печать начинается с новой страницы, страницы начинают нумероваться заново арабскими цифрами. Наконец, команда `\backmatter` задает переход к заключительной части книги: главы перестают нумероваться (нумерация страниц не меняется).

5.5. Перемещаемые аргументы и хрупкие команды

Если в аргументе команды `\section` (или любой другой LaTeX'овской команды для рубрикации) присутствует не только текст, но и TeX'овские команды, то при трансляции они могут иногда вызвать сообщение об ошибке. Чтобы этого избежать, команду надо «защитить»: поставить непосредственно перед ней команду `\protect`. Приведем пример, когда возникает нужда в этой команде.

Предположим, вы решили сказать LaTeX'у, что в каком-то месте заголовок раздела нельзя разрывать на печати, с помощью команды `\nolinebreak`. Тогда надо действовать следующим образом:

```
\documentclass{article}

\begin{document}
\section{Устойчивость по Ляпунову}
     ...
\section{О некоторых свойствах функций\protect\nolinebreak\
Ляпунова}
Еще кое-что.
\end{document}
```

(Заметьте заодно, что команда `\nolinebreak` дана до пробела между словами, а не после — иначе она вообще не сработает, невзирая ни на какой `\protect`. См. с. 123.) Если убрать в этом файле `\protect`, то на экране появится загадочное сообщение об ошибке.

Такого рода ситуация может возникать, когда TeX'овская команда является частью текста, который будет записан в специальный файл и использован при следующем запуске LaTeX'а (в нашем случае информация о заголовке раздела записывается в файл с расширением `toc` для последующего использования в оглавлении). Если аргумент команды (в нашем случае команды `\section`) подвергается такой обработке, то его называют *перемещаемым*; команды, которые, будучи использованы внутри перемещаемого аргумента, могут вызвать ошибку, называются *хрупкими*.

Из тех LaTeX'овских команд, которые могут реально понадобиться внутри заголовка раздела, большинство хрупкими не являются. Если вы сомневаетесь, не хрупка ли какая-то конкретная команда, можете спокойно ставить перед ней `\protect` — ничего плохого от этого не произойдет.

Не являются хрупкими и не нуждаются в защите с помощью `\protect` команды для расстановки диакритических знаков (см. с. 106), смены текущего шрифта и установки промежутков вручную (см. с. 106).

6. Титульный лист, оглавление, список литературы, аннотация

6.1. Титульный лист

Для того, чтобы оформить заголовок ко всему документу, надо сделать две вещи: задать информацию для заголовка (автор, название и т. п.) и дать LaTeX'у команду этот заголовок сгенерировать. Второе делается с помощью команды `\maketitle`. Она создаст титульный лист, если это предусмотрено классом и опциями. (Если титульный лист не предусмотрен, то команда `\maketitle` разместит заданную вами информацию об авторе, заглавии и прочем на первой странице, выбрав подходящие шрифты и сделав подобающие отступы между титульной информацией и текстом.) По умолчанию для классов `report` и `book` титульный лист создается всегда (и не создается, если указана классовая опция `notitlepage`), для класса `article` титульный лист не создается (но будет создан, если указать классовую опцию `titlepage`). В классе `proc` титульная информация всегда печатается на первой странице текста.

Так как команда `\maketitle` генерирует текст, ее нельзя помещать в преамбуле документа.

Теперь объясним, как задавать LaTeX'у информацию для титула. Автор задается с помощью команды `\author`. Она принимает единственный обязательный аргумент — имя автора (в том виде, как вы хотите его видеть на титуле). Если авторов несколько, их имена должны быть разделены командой `\and`.

Заглавие задается с помощью команды `\title`. Если заглавие длинное, можно самому задать его разбиение на строки с помощью команды `\\`; если этого не сделать, заглавие будет разбито на центрированные строки автоматически, как если бы это был абзац в окружении `center` (см. с. 129, а также пример на с. 25).

Следующий элемент информации для титула — команда `\date`. Она имеет один обязательный аргумент, в котором можно задать любой текст (например, дату, в согласии с переводом слова date), который будет размещен на титульном листе (или перед началом основного текста, если титульный лист не предусмотрен классом и/или опциями) в одной или нескольких центрированных строках (так же, как и текст, задаваемый в аргументе команды `\title`). В частности, можно оставить аргумент этой команды «пустым», если сказать `\date{}` — тогда соответствующий текст вообще не появится. Но если вы вообще не дадите эту команду, хотя бы и с пустым аргументом, то LaTeX напечатает на титуле дату своего запуска, причем по-английски[2].

Команды `\author`, `\title` и `\date` можно давать в любом порядке, но обязательно до команды `\maketitle` (можно и в преамбуле). Команда `\maketitle` должна быть первой из команд, генерирующих текст.

Наконец, последнее, что можно сделать с информацией для титула документа, — это снабдить ее сносками. К любому из авторов, к любым словам в титуле или в тексте, содержащемся в аргументе команды `\date`, можно сделать сноску с помощью команды `\thanks`, имеющей один обязательный аргумент — текст сноски (в отличие от обычных сносок, абзацы в этом тексте нельзя разделять пустыми строками или командами `\par`; если в вашей сноске должно быть несколько абзацев, разделяйте их TeX'овской командой `\endgraf`).

[2] Или по-русски, если это предусмотрено в используемой вами русификации.

Сноски будут напечатаны внизу титульного листа (или первой страницы, если титульный лист не предусмотрен). Пример:

```
\author{Борис Заходер}
\title{Винни-Пух и все-все-все\thanks{Вообще-то
это перевод из A.\,A.\,Милна}}
\date{}
```

Обратите внимание, что команда `\thanks` помещается *внутри* аргумента команд `\title` и/или `\author`.

Наконец, можно при желании вообще не использовать стиль оформления титульного листа, диктуемый нам LaTeX'ом. Сделать это очень просто — надо воспользоваться окружением `titlepage`. Текст между `\begin{titlepage}` и `\end{titlepage}` составит титульный лист, за оформление которого целиком отвечает тот, кто текст готовит. Сам LaTeX внутри этого окружения делает только три вещи:

- устанавливает печать в одну колонку (даже если сам документ будет печататься в две колонки);
- начинает новую страницу и устанавливает счетчик числа страниц в нуль;
- устанавливает странице стиль оформления `empty` (без колонтитула и номера).

Что и как разместить на этой странице — ваша забота.

6.2. Оглавление

В процессе работы LaTeX автоматически собирает информацию для создания оглавления и записывает ее в специальный файл с тем же именем, что у обрабатываемого файла, и расширением `toc`. Чтобы LaTeX записал эту информацию, а затем воспользовался ею и напечатал оглавление, надо дать команду `\tableofcontents`.

Стало быть, оглавление, генерируемое LaTeX'ом, всякий раз будет «на шаг отставать» от реального положения дел. Чтобы учесть все возможные изменения и получить верное оглавление, надо будет в самом конце работы над текстом запустить LaTeX еще раз (напоминания об этом LaTeX не выдаст).

Все оглавление в целом будет озаглавлено словом, определяемым командой `\contentsname` (см. разд. 5.3).

Если вас не устраивает стандартный стиль оформления оглавления, прочтите в последней главе, как его можно изменить.

6.3. Список литературы

Имеется возможность оформить список литературы, элементы которого нумеруются автоматически; в тексте при этом надо ссылаться не на эти номера, которые могут измениться в процессе работы над документом, а на установленные вами условные обозначения для элементов списка литературы («источников»).

Список литературы оформляется как окружение `thebibliography`. Это окружение имеет обязательный аргумент — номер источника, занимающий больше всего места на печати (в стандартных шрифтах все цифры имеют одинаковую ширину, так что достаточно привести в качестве аргумента, например, номер 99, если источников будет заведомо меньше 100).

Каждый источник вводится командой `\bibitem`. У нее есть один обязательный аргумент — ваше условное обозначение. В качестве такого обозначения можно использовать любую последовательность из букв и цифр.

В тексте ссылка на источник делается с помощью команды `\cite`. У нее есть обязательный аргумент — условное обозначение того источника, на который вы ссылаетесь. Можно сослаться сразу на несколько источников — для этого в аргументе команды `\cite` надо указать их обозначения через запятую. Приведем пример (в котором для экономии места мы опустили заголовок «Список литературы»):

В [3, гл. 1] описана встреча Винни-Пуха с несколькими пчелами. В [1, 2] приведены другие сведения о медведях.

[1] М. Е. Салтыков-Щедрин. Медведь на воеводстве.

[2] Л. Н. Толстой. Три медведя.

[3] А. А. Милн. Винни-Пух.

```
В~\cite[гл.~1]{Winnie}
описана встреча Винни-Пуха
с несколькими пчелами.
В~\cite{voevoda,med3}
приведены другие
сведения о медведях.
\begin{thebibliography}{99}
\bibitem{voevoda}
М.\,Е.\,Салтыков-Щедрин.
Медведь на воеводстве.
\bibitem{med3} Л.\,Н.\,Толстой.
Три медведя.
\bibitem{Winnie}
А.\,А.\,Милн. Винни-Пух.
\end{thebibliography}
```

В этом примере вы также можете видеть команду `\cite` с необязательным аргументом: он ставится перед обязательным; в квадратных скобках записывается текст, который будет через запятую напечатан после номеров ссылок.

Как это обычно и происходит с автоматически генерируемыми LaTeX'ом ссылками, после первого запуска программы вы увидите сообщение о том, что ссылки не определены. Если в дальнейшем в процессе работы над текстом нумерация ссылок изменится, LaTeX сообщит вам об этом и предложит запустить программу еще раз, чтобы получить корректные ссылки.

Если вам не нравится, что источники в списке литературы нумеруются, можно придумать для них свои обозначения, которые будут печататься вместо номеров. Для этого надо использовать команду `\bibitem` с необязательным аргументом, идущим перед обязательным. В квадратных скобках ставится то обозначение, которое будет заменять номер для этого источника. Например, можно написать так:

```
\begin{thebibliography}{XXXX}
...
\bibitem[EGA]{Groth} A.\,Grothendieck, J.\,Dieudonn\'e.
\'El\'ements de G\'eom\'trie Alg\'ebrique.
...
\end{thebibliography}
```

После этого команда `\cite{Groth}` будет генерировать текст [EGA].

Списку литературы в целом LaTeX автоматически дает заглавие, определяемое командой `\refname` в классах `article` и `proc` и `\bibname` в классах `report` и `book` (см. разд. 5.3). Если это заглавие печатается по-английски, его можно переопределить (см. там же).

7. Предметный указатель

В отличие от списка литературы, который при использовании описанных выше команд `\cite` и `\bibitem` получается совершенно автоматически, процесс создания указателя автоматизирован в LaTeX'е не полностью. Именно, вы можете сделать две вещи:

- Если вам уже известно, какие термины должны войти в указатель и на каких страницах они расположены, можно организовать печать предметного указателя с помощью окружения `theindex`. Если предметный указатель должен завершать документ, то можно, на худой конец, напечатать весь документ, кроме указателя, и вручную выписать требуемые номера страниц[3].

[3] У вас может возникнуть искушение пометить все места, на которые надо сослаться, с помощью команды `\label`, а в окружении `theindex` получить номера страниц с помощью `\pageref`. При этом, однако, есть опасность, что TeX'у не хватит памяти для обработки такого огромного числа ссылок.

- Можно также (и на практике это удобнее) специальным образом пометить в файле термины, на которые вы собираетесь ссылаться в предметном указателе. При этом средствами LaTeX'а создается полуфабрикат, из которого предметный указатель получится после обработки отдельной программой (называемой обычно `makeindex`, а в русской версии — `rmakeindex` или как-нибудь еще в этом роде), входящей в настоящее время почти во все поставки LaTeX'а.

Начнем с того, что расскажем о втором способе создания индекса, а потом объясним, что можно сделать, если программа для генерации индекса вам по какой-то причине недоступна.

7.1. Общие положения

Чтобы разметить файл для автоматической генерации индекса, нужно сделать две вещи. Во-первых, в преамбулу документа необходимо включить команду `\makeindex`. Во-вторых, при условии, что это сделано, можно пометить те места в тексте, на которые вы хотите сослаться в предметном указателе, командой `\index` (если команда `\makeindex` в преамбуле отсутствует, то команды `\index` ничему не мешают, но и никакого действия не оказывают). У этой команды один обязательный аргумент — текст вашей пометки (в простейшем случае такая пометка — это ключевое слово будущего предметного указателя). Информация о том, на какие страницы попали ваши пометки, будет записана в специальный файл с тем же именем, что и у вашего файла, и расширением `idx` (мы будем называть его `idx`-файлом). Пусть, например, в исходном файле встречались такие фрагменты[4]:

```
Многие люди любят домашних кошек.\index{Кошки}
....
Хорошо также иметь собаку.\index{Собаки}
....
Мало кто рискнет держать дома такую дикую
кошку,\index{Кошки} как тигр.
```

[4]В наших примерах мы используем русские буквы, хотя на самом деле использование русских букв в предметном указателе зависит от применяемой вами русификации. Прежде всего, в ее состав должна входить версия программы `makeindex`, учитывающая русский алфавитный порядок. Кроме того, иногда (например, при использовании русского варианта пакета `babel`) в `idx`-файл попадают не русские буквы, а их условные обозначения, а в других случаях (при использовании старых версий TeX'а) их шестнадцатеричные коды, и это создает дополнительные сложности. Возможны также сложности с буквой «ё» при сортировке. При подготовке этой книги использован вариант русификации (см. приложение E), где такие проблемы не возникают.

Предположим, что первая ссылка на кошек попала на страницу 5, ссылка на собак попала на страницу 7, а вторая ссылка на кошек попала на страницу 9. Тогда в `idx`-файл запишется вот что:

```
\indexentry{Кошки}{5}
\indexentry{Собаки}{7}
\indexentry{Кошки}{9}
```

Полученный таким образом `idx`-файл — это и есть полуфабрикат указателя, созданный LaTeX'ом. Использовать этот полуфабрикат, однако же, еще нельзя: ссылки в `idx`-файле расположены не по алфавиту, а записаны «в порядке поступления», в `idx`-файле может присутствовать несколько строк с одним заглавным словом и ссылками на разные страницы, наконец, команда `\indexentry`, с которой начинается каждая строка `idx`-файла, не определена в LaTeX'е (это сделано сознательно!).

Поэтому, получив `idx`-файл, надо его обработать с помощью программы `makeindex`; в результате получится файл с отсортированными по алфавиту терминами (обычно он имеет расширение `ind` и называется `ind`-файлом), который можно будет включить в окружение `theindex`, написав

```
\begin{theindex}
\input{text.ind}
\end{theindex}
```

Если вы заглянете в файл, полученный в результате работы программы `makeindex`, то увидите, что в окружении `theindex` каждый элемент указателя вводится командой `\item`; команды `\subitem` и `\subsubitem` вводят элементы указателя, печатающиеся с дополнительными отступами (обычно это уточнения к заглавному слову) — вскоре мы объясним, что надо писать в аргументе команды `\index`, чтобы получить такую иерархию элементов указателя. Наконец, команда `\indexspace` создает дополнительный вертикальный пробел (его можно использовать для отделения различных разделов указателя друг от друга):

Компьютеры, 25–42
    IBM-совместимые, 28
      ремонт, 35
      цены, 30
    болгарские, 26
Принтеры, 40

Кошки, 120
Собаки, 140–156

```
\begin{theindex}
\item Компьютеры, 25--42
\subitem IBM-совместимые, 28
\subsubitem ремонт, 35
\subsubitem цены, 30
\subitem болгарские, 26
\item Принтеры, 40
\indexspace
\item Кошки, 120
\item Собаки, 140--156
\end{theindex}
```

Предметный указатель, получаемый из окружения `theindex`, печатается LaTeX'ом в две колонки (даже тогда, когда сам документ печатается в одну колонку). Кроме того, LaTeX автоматически дает указателю заглавие, определяемое командой `\indexname` (см. разд. 5.3). Если вас не устраивает, что это название — английское, его можно переопределить (см. там же).

В аргументе команды `\index` могут быть любые символы, и вообще текст в аргументе этой команды может быть неосмысленным или недопустимым с точки зрения TeX'а — в любом случае аргумент команды `\index` будет в неизменном виде переписан в `idx`-файл. Смысл тут в том, что в аргументе команды `\index` можно задавать вспомогательную информацию для программы обработки `idx`-файла (примеры тому вы найдете ниже). Единственное ограничение — не должно быть «несбалансированных» фигурных скобок, даже если эти скобки входят в состав команд `\{` или `\}` (напомним, что вместо `\{` или `\}` всегда можно написать `\lbrace` или `\rbrace` соответственно).

Наконец, еще одна тонкость: команду `\index` нельзя использовать внутри необязательного аргумента таких команд, как `\section`, `\chapter`, `\caption` (подрисуночная подпись; см. следующий раздел) и т. п.

7.2. Простейшие средства

В простейшем случае программа `makeindex` вызывается так[5]:

`makeindex` *исходный_файл*

Если *исходный_файл* имеет расширение `idx` (так скорее всего и будет, поскольку исходный файл — это, как правило, сгенерированный LaTeX'ом `idx`-файл), то это расширение можно не указывать. В результате работы программы `makeindex` появится файл с тем же именем, что у исходного файла, и расширением `ind`. Это — готовый файл для предметного указателя, который остается только включить в ваш документ с помощью команды `\input`. Создается также файл с тем же именем и расширением `ilg`. Это — протокол работы программы `makeindex`.

Если не предпринимать специальных мер, то все записи в `ind`-файле, созданном программой `makeindex`, будут равноправны — все они будут вводиться командой `\item`. Чтобы предметный указатель был устроен иерархически, как в примере на с. 178, надо в аргументе команды `\index` после заглавного слова поставить восклицательный знак, а после него — подчиненное ему слово. Возможно также подчинение второго порядка — тогда нужен еще один восклицательный знак. Вот пример:

[5]Напомним, что конкретное название программы и возможность использования русских букв может зависеть от используемой системы и русификации. См. приложения Е и Ж.

```
Многие люди любят домашних кошек.\index{Кошки!домашние}
....
Ваша киска\index{Кошки!домашние!уход} купила бы...
....
Хорошо также иметь собаку.\index{Собаки}
.....
Мало кто рискнет держать дома такую дикую
кошку,\index{Кошки} как тигр. Пудель\index{Собаки}
гораздо безопаснее.
```

При обработке этого файла LaTeX'ом получится `idx`-файл[6]; в результате обработки `idx`-файла программой `makeindex` получится `ind`-файл, включающий в себя, в частности, следующее (предположим, что наши команды `\index` попали на страницы с номерами 2, 7, 8, а две последние — на страницу 9):

```
\begin{theindex}
...
  \item Кошки, 9
    \subitem домашние, 2
      \subsubitem уход, 7
...
  \item Собаки, 8, 9
...
\end{theindex}
```

Из сказанного следует, что при обработке `idx`-файла с помощью программы `makeindex` восклицательный знак в аргументе команд `\index` имеет особый статус. Чтобы программа `\makeindex` восприняла восклицательный знак просто как типографский значок, надо в аргументе `\index` предварить его знаком кавычки `"`:

```
\index{Восклицательный знак ("!)}
\index{Междометия!Эх"!}
```

Эти аргументы команд `\index` дословно скопируются в `idx`-файл, а после его обработки программой `makeindex` в `ind`-файл запишется примерно вот что:

```
\item Восклицательный знак (!), 14
\item Междометия
  \subitem Эх!, 6
```

[6] В том, конечно, случае, если в преамбуле была команда `\makeindex`.

Наряду с восклицательным знаком, особый статус с точки зрения программы `makeindex` имеет символ `@` («коммерческое at»), вертикальная черточка `|` (в следующем разделе вы узнаете, в чем этот статус заключается), а также сама кавычка `"`. Если вы хотите употребить один из этих четырех значков в аргументе команды `\index` просто как символ, не вкладывая в него специального смысла, надо поставить перед ним кавычку `"`.

Исключение: кавычку, входящую в состав TEX'овской команды `\"`, можно (и нужно) записывать без предосторожностей:

```
\index{Кавычка ("")}
\index{Ёлочка}
\index{H\"uhnchen}
```

При этом соответствующие записи в `idx`-файле могут получиться такими:

```
\item Ёлочка, 8
\item Кавычка ("), 6
\item H\"uhnchen, 12
```

7.3. Тонкости

Для каждого ключевого слова программа `makeindex` собирает все относящиеся к нему номера страниц и записывает их в `ind`-файле после этого слова через запятую. Если при этом попадутся три или более идущих подряд номера страниц, то в `ind`-файл будут записаны только первый и последний из этих номеров, через короткое тире (en-dash). Такие пары страниц через короткое тире можно организовывать и вручную. Пусть, например, в какой-то части вашего текста все время идет речь о кошках. Тогда можно в начале этой части написать

```
\index{Кошки|(}
```

а в конце —

```
\index{Кошки|)}
```

Если первая из этих команд попала на страницу 9, а вторая — на страницу 77, то после обработки `idx`-файла программой `makeindex` в `ind`-файл попадет запись

```
\item Кошки, 9--77
```

Команды `\index{Кошки}`, оказавшиеся между страницами 9 и 77, будут при этом проигнорированы.

При сортировке программа `makeindex` принимает во внимание не только буквы, но и спецзнаки, записанные в аргументе команды `\index`. Иногда это нежелательно: если в тексте имеются команды `\index{Аист}` и `\index{\textbf{Ящерица}}`, то ящерица может оказаться в `ind`-файле раньше аиста, если `makeindex` будет считать, что запись для нее начинается с символа \, который идет раньше всех русских букв[7]. Чтобы избежать такого рода неприятностей, предусмотрена возможность по отдельности задать слово, которое будет участвовать в сортировке, и текст, который будет реально записан в `ind`-файл. В приведенном выше примере следовало бы написать

```
\index{Ящерица@\textbf{Ящерица}}
```

Теперь ящерица попадет туда же, куда и все прочие слова на букву «я», но при этом будет напечатана жирным шрифтом. Общее правило такое: если в аргументе команды `\index` присутствует символ `@`, то при сортировке учитывается только то, что написано *левее* него, а в `ind`-файл записывается только то, что *правее* него. Можно задавать отдельные тексты для сортировки и для печати не только для основного заглавного слова, но и для слов, ему подчиненных:

```
\index{Ящерицы@\textbf{Ящерицы}!игуана@\textbf{игуана}}
```

Напоминание: если перед `@` или `|` стоит кавычка `"`, то эти значки рассматриваются просто как символы.

Программа `makeindex` может оформлять номера разных страниц поразному. Пусть, например, вы считаете, что одно из мест в тексте, где говорится о кошках, является особо важным, и хотите, чтобы номер соответствующей страницы был подчеркнут (по аналогии с указателем к книге TeXbook [2]). Тогда можно поставить в этом месте команду

```
\index{Кошки|underline}
```

Предположим, что эта команда попала на страницу 100, и, кроме того, в тексте были две команды `\index{Кошки}`, попавшие на страницы 15 и 47. Тогда после обработки `idx`-файла программой `makeindex` в `ind`-файле появится такая строка:

```
\item Кошки, 15, 47, \underline{100}
```

[7] Он может идти и позже — в зависимости от конкретной версии программы `makeindex` и ее отношения к русским буквам.

Общее правило таково: команда `\index{XXX|abcd}` порождает в `ind`-файле строку

`\item XXX, \abcd{`y`}`

(здесь y — номер страницы).

Если вы хотите номер страницы не подчеркнуть, а выделить другим шрифтом, разумно воспользоваться одной из команд для смены шрифта, работающих как команда с одним аргументом (см. разд. III.5.2).

Среди символов `abcd` не должно быть круглых скобок (сочетания `|(` и `|)` имеют, как было сказано выше, особый смысл); если вы хотите оформить таким образом номер страницы к «подчиненному», а не заглавному термину, то именно после подчиненного термина (того, после которого должен реально появиться номер страницы) и надо писать `|abcd`:

```
\index{Кошки!Ангорские|textbf}
```

7.4. Настройка программы makeindex

В предыдущих разделах мы объясняли, как и какую информацию можно передавать программе `makeindex`. Теперь объясним, как добиться того, чтобы она обрабатывала эту информацию по-иному.

Если «запустить программу `makeindex` с ключом `c`», то есть сказать

`makeindex -c` *исходный_файл*

то при обработке `idx`-файла программой `makeindex` пробелы в начале и конце записей будут игнорироваться, а два и более пробела будут рассматриваться как один. Благодаря этому записи `\index{Кошка}` и `\index{ Кошка}` будут рассматриваться как относящиеся к одному и тому же ключевому слову. Без этого ключа программа `makeindex` отведет в указателе отдельную строку для кошки, начинающейся с пробела.

Можно задать по своему усмотрению имя файла, в который программа `makeindex` запишет результаты своей работы. Для этого надо воспользоваться ключом `o` (в примере мы употребили еще и ключ `c`, но это не обязательно):

`makeindex -c -o` *выходной_файл исходный_файл*

Чтобы задать отличное от стандартного имя файла с протоколом трансляции, надо аналогичным образом воспользоваться ключом `t`.

Наконец, можно запустить программу `makeindex` вместе со *стилевым файлом,* в котором программе будут даны указания по поводу вида, в котором будет записан отсортированный и обработанный `idx`-файл.

Написав подходящий стилевой файл для `makeindex`, можно радикально изменить вид `ind`-файла — так, что он вообще не будет иметь ничего общего с LaTeX'овским файлом (это может иметь смысл: `makeindex` задуман как программа широкого профиля, пригодная не только для LaTeX'а). Как добиться столь революционных изменений, мы обсуждать не будем (интересующиеся могут узнать все подробности из оригинальной документации), но о некоторых вещах, полезных именно для LaTeX'а, расскажем.

Чтобы подключить стилевой файл к `makeindex`, надо запустить эту программу с ключом `s`, после которого, через пробел, указывается имя стилевого файла (по традиции он имеет расширение `ist`). Если стилевой файл называется `mystyle.ist`, то можно сказать так:

`makeindex -s mystyle.ist` *исходный_файл*

Теперь обсудим, что можно менять с помощью стилевого файла. Как мог заметить читатель, программа `makeindex` автоматически записывает строку

```
\begin{theindex}
```

в начало `ind`-файла и

```
\end{theindex}
```

в его конец. Часто требуется, чтобы в начало или конец `ind`-файла автоматически записывалось что-то еще (команда `\sloppy` в начало, например). Для того, чтобы после `\begin{theindex}` было на отдельной строке написано еще и `\sloppy`, надо в стилевом файле написать так:

```
preamble "\\begin{theindex}\n
        \\sloppy\n"
```

Здесь `preamble` — имя стилевого параметра, определяющего, что записывается в начало всякого `ind`-файла. Остальной текст — содержание этой записи. Правила записи в стилевом файле для `makeindex` таковы:

- строковая константа, задающая стилевой параметр, ограничена с обеих сторон знаками `"` (кавычки);
- эта строковая константа может реально состоять и из нескольких строк; место, где кончается одна строка и начинается другая, обозначается `\n` (конец строки воспринимается просто как пробел и не означает конца строки в `ind`-файле);

- если в строковую константу должны входить символы `\` или `"`, то их надо обозначать `\\` и `\"` соответственно, а все остальные символы набираются непосредственно.

Параметр `postamble` определяет, что записывается в конец `ind`-файла. По умолчанию это

```
"\n\n\\end{theindex}\n"
```

(иными словами: начать с новой строки, одну строку пропустить, написать `\end{theindex}`, строку закончить).

Следующие три параметра определяют, чем отделяются номера страниц от ключевых слов: `delim_0` — для ключевых слов «верхнего уровня», `delim_1` и `delim_2` — для слов первого и второго уровня подчинения. По умолчанию все три этих параметра определены как `", "` (запятая и пробел), вследствие чего номера страниц отделяются от слов запятыми. В русских текстах эти запятые ставить не принято, поэтому все три этих параметра стоит переопределить на `" "`:

```
delim_0 " "
delim_1 " "
delim_2 " "
```

Параметр `group_skip` определяет, что записывается в `ind`-файл между группами слов, начинающихся на одну букву. Значение по умолчанию — `"\n\n \\indexspace\n"` («пропустить строку, написать слово `\indexspace` и начать с новой строки»).

Перед каждой группой терминов, начинающихся с новой буквы, можно (как это и сделано в указателе к книге, которую вы держите в руках) напечатать на отдельной строке эту букву. Для этого в стилевом файле надо написать

```
headings_flag 1
```

При этом буквы будут напечатаны тем же шрифтом, что и остальной текст указателя. Чтобы шрифт был другим, надо в стилевом файле определить параметры `heading_prefix` и `heading_suffix`. Первый из них определяет, какой TEX'овский текст запишется в `ind`-файл перед буквой, второй — что запишется после буквы. Если, например, нужно, чтобы шрифт, которым печатаются буквы-заголовки, имел размер `\large` и был полужирным, то можно написать

```
heading_prefix "{\\normalfont\\large\\bfseries "
heading_suffix "}"
```

(вспомните, что для получения символа \ надо написать \\; пробел после bfseries необходим, чтобы в ind-файле имя команды не слилось с последующей буквой).

Если в указателе присутствуют термины, начинающиеся с символа, не являющегося буквой, то они будут выделены в отдельную группу; если в стилевом файле написано headings_flag 1, то перед этой группой будет напечатано слово Symbols. Если вас это не устраивает, надо в стилевом файле определить строковую константу symhead_positive. Если, например, написать

```
symhead_positive ""
```

то перед этой группой вообще никакого заголовка не будет; если хотите, чтобы вместо Symbols был другой заголовок, напишите вместо "" этот заголовок (в двойных кавычках, разумеется).

7.5. Если программы makeindex нет

Теперь рассмотрим вторую возможность — как самостоятельно обработать автоматически созданный с помощью LaTeX'а полуфабрикат предметного указателя.

Если не писать самостоятельно программу для обработки idx-файлов, то можно, по крайней мере, сделать следующее.

Во-первых, надо отсортировать строки idx-файла (средствами текстового редактора, например). После этого остается проблема, что делать с командами \indexentry. После того, как вы отсортируете idx-файл (и сохраните, для надежности, отсортированный файл под другим именем, скажем, myindex.tex), надо определить команду \indexentry таким образом, чтобы она делала ту же работу, которую призван делать \item. Для этого надо написать в преамбуле следующее:

```
\newcommand{\indexentry}[2]{\item #1 #2}
```

После этого TeX будет воспринимать каждую запись вида

```
\indexentry{Кошки}{5}
```

так же, как если бы вместо этого было написано

```
\item Кошки 5
```

и можно будет просто написать в конце документа

```
\begin{theindex}
\input{myindex.tex}
\end{theindex}
```

Пока что воспринимайте этот рецепт чисто догматически; по прочтении гл. VII, в которой подробно рассмотрен процесс определения новых команд, вы поймете, почему этот рецепт работает.

8. Плавающие иллюстрации и таблицы

8.1. Простейшие средства

Чтобы разместить в тексте иллюстрацию, удобно воспользоваться окружением `figure`. Стоящий между `\begin{figure}` и `\end{figure}` текст автоматически размещается LaTeX'ом в таком месте, где он укладывается целиком (не переходя со страницы на страницу); это может быть не на «своей» странице, а позже. В последнем случае говорят, что иллюстрация «всплыла» на следующей странице; именно поэтому окружение `figure` называют еще «плавающая иллюстрация».

Команда `\caption` позволяет сделать подрисуночную подпись. Эта команда имеет один обязательный аргумент — текст подписи. На печати подпись состоит из слова, определенного командой `\figurename` («Figure», если не переопределять эту команду — см. разд. 5.3), порядкового номера иллюстрации, присвоенного ей LaTeX'ом, и подписи, указанной в аргументе команды. Команду `\caption` можно давать в любом месте между `\begin{figure}` и `\end{figure}`: в соответствующем месте появится на печати и сгенерированная ею подпись. Разумно поэтому ставить команду `\caption` либо в конце окружения `figure` (тогда подпись будет размещена под иллюстрацией), либо в его начале (подпись появится над иллюстрацией).

Если команду `\label` поместить внутри окружения `figure` после команды `\caption`, то команда `\ref` будет генерировать номер иллюстрации. Например, рис. IV.3 на с. 187 получился так:

```
Например, рис.~\ref{void} на с.~\pageref{void} ...
\begin{figure}
\vspace*{2.5cm}
\caption{Белый квадрат
на белом фоне}\label{void}
\end{figure}
```

Внутри одного окружения `figure` может быть несколько команд `\caption`, и каждую из них можно пометить своей командой `\label`. Подчеркнем, что само по себе окружение `figure` номера рисунка не создает.

На подписи к рис. IV.3 после номера стоит точка. К сожалению, согласно LaTeX'овскому стандарту в этом месте ставится не точка, а двоеточие, что в русском тексте выглядит неудачно. В главе IX объясняется, как с этим можно бороться.

У окружения `figure` предусмотрен необязательный аргумент, с помощью которого можно высказать LaTeX'у свои пожелания по поводу размещения иллюстрации в тексте. Именно, после `\begin{figure}` (без

Рис. IV.3. Белый квадрат на белом фоне

пробела) можно поместить в квадратных скобках одну или несколько из следующих четырех букв, имеющих такие значения:

- `t` разместить иллюстрацию в верхней части страницы;
- `b` разместить иллюстрацию в нижней части страницы;
- `p` разместить иллюстрацию на отдельной странице, целиком состоящей из «плавающих» иллюстраций (или таблиц — см. ниже);
- `h` разместить иллюстрацию прямо там, где она встретилась в исходном тексте, не перенося ее никуда.

Если в квадратных скобках стоит несколько букв, это значит, что вы согласны на любой из предусматриваемых этими буквами вариантов. Если окружение `figure` задано без необязательного аргумента, это равносильно записи

```
\begin{figure}[tbp]
```

При наборе текста в две колонки полезно использовать не только само окружение `figure`, но и его вариант «со звездочкой» (см. разд. I.2.8): если сказать

```
\begin{figure*}
```

то при наборе текста в одну колонку это не будет ничем отличаться от окружения `figure` без звездочки, а при наборе текста в две колонки создаст иллюстрацию шириной в целую страницу (без звездочки получилось бы шириной в одну колонку). Если окружение открывается командой `\begin{figure*}`, то и закрываться оно должно командой со звездочкой.

Если при наборе в две колонки задать окружение `figure` (без звездочки) с необязательным аргументом `p`, то для печати иллюстраций будет выделена не отдельная страница, но отдельная колонка. При подключении стилевого пакета `multicol` пользоваться окружениями со звездочкой `figure*` и `table*` (см. ниже) нельзя.

Окружение `table` определяет «плавающие таблицы». Все свойства этого окружения дословно совпадают с соответствующими свойствами окружения `figure`, за двумя исключениями: во-первых, подпись, генерируемая командой `\caption`, начинается со слова, определенного в команде `\tablename` (см. разд. 5.3), и переопределять, при необходимости, надо именно эту команду, и во-вторых, таблицы нумеруются независимо от иллюстраций. Кстати, подпись к таблице принято делать не снизу, как к иллюстрации, а сверху. Окружение `table*` при наборе текста в две колонки определяет таблицы шириной в целую страницу.

В документе можно, при желании, получить автоматически сгенерированные списки иллюстраций и/или таблиц. Для этого используются команды `\listoffigures` (для иллюстраций) и `\listoftables` (для таблиц). Их работа аналогична команде `\tableofcontents`, генерирующей оглавление (см. с. 173): материал для этих списков собирается в специальные файлы с расширениями `lof` (для иллюстраций) и `lot` (для таблиц); при каждом запуске LaTeX'а информация, записанная в этих таблицах, относится к предыдущему запуску, так что в самом конце может понадобиться запустить LaTeX лишний раз; наконец, команда `\caption` может принимать необязательный аргумент — вариант подписи под иллюстрацией или таблицей, предназначенный для включения в список иллюстраций или таблиц соответственно. Этот необязательный аргумент записывается (в квадратных скобках, как обычно) *перед* обязательным.

Как окружение `figure` не рисует картинок, так и окружение `table` только размещает таблицу на страницах документа, но не создает ее текста. Как набирать таблицы в LaTeX'е, мы расскажем в гл. VI.

В разд. IX.7 мы расскажем о том, как можно модифицировать оформление плавающих иллюстраций и таблиц.

8.2. Рисунки в оборку

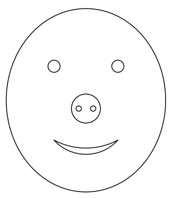


Рис. IV.4.

Окружения `figure` и `table` определяют иллюстрации и таблицы, простирающиеся на всю ширину текста, и ничего иного в стандартном комплекте LaTeX'а не предусмотрено. Пользователями LaTeX'а разработано несколько стилевых пакетов, позволяющих с бо́льшим или меньшим успехом печатать прямоугольные иллюстрации, обтекаемые текстом. В этом разделе мы расскажем о возможностях, предоставляемых стилевым пакетом `wrapfig`. Этот пакет, разработанный Дональдом Арсено (Donald Arseneau), довольно удобен на практике (по крайней мере, автор книги и его коллеги не без успеха им пользуются), хотя, конечно, размещение обтекаемых текстом иллюстраций полностью автоматизировано быть не может и всегда требует определенной ручной работы. Ниже мы опишем основные принципы использования пакета `wrapfig`, но реально пользоваться им вы сможете только после того, как освоите следующую главу о псевдорисунках или (что лучше) приложение Б, посвященное интеграции в TeX графических файлов в формате PostScript.

Итак, предположим, что стилевой пакет `wrapfig` подключен. Тогда рисунок, обтекаемый текстом, надо задать как окружение `wrapfigure` (в том же стилевом пакете определено окружение `wraptable`, задающее обтекаемую текстом таблицу; аргументы у этого окружения имеют в точности такой же смысл, как у окружения `wrapfigure`).

Окружение `wrapfigure` имеет два обязательных аргумента. Первый из них указывает, как должен быть расположен рисунок относительно текста, а второй — ширину рисунка (заданную в TeX'овских единицах длины или выраженную через TeX'овские параметры со значением длины). Например, код, задающий рис. IV.4, мог выглядеть примерно так:

```
\begin{wrapfigure}{o}{62.2pt}
    ⟨команды, задающие рисунок⟩
\end{wrapfigure}
```

Латинская буква `o` в первом аргументе означает, что рисунок должен быть расположен на наружной (outer) стороне страницы (то есть справа на нечетных страницах и слева — на четных); если бы мы сказали `i` вместо `o`, то рисунок был бы расположен на внутренней стороне страницы. Впрочем, оба эти аргумента имеют смысл только в том случае, если документ является «двусторонним» (с классовой опцией `twoside`; см. с. 157, 163); если же набор «односторонний», то в первом обязательном аргументе надо указать букву `l` (рисунок слева) или `r` (справа).

Каждая из букв `o`, `i`, `l` и `r` может быть не только строчной, но и прописной; в этом случае рисунок при необходимости может быть помещен не буквально там, где в исходном тексте находится окружение `wrapfigure`, но передвинут в другое место.

Количество укороченных строк, необходимое для обтекания рисунка текстом, LaTeX рассчитывает самостоятельно; если обтекающий текст содержит команды для явного задания вертикальных промежутков или занимающие много места по вертикали выключные формулы, результат такого расчета может быть неверен. На этот случай у окружения `wrapfigure` предусмотрен необязательный аргумент, ставящийся *перед первым обязательным* — количество укороченных строк. Пример:

```
\begin{wrapfigure}[14]{o}{60pt}
```

Имейте в виду, что при таких расчетах любая выключная формула считается за три строки.

Идеальное место для размещения окружения wrapfigure — между абзацами. Если вам нужно поместить обтекаемую иллюстрацию внутри абзаца, посмотрите, где TEX сделает в этом абзаце разрывы строк, если иллюстрацию не помещать, и начинайте окружение wrapfigure после слова, заканчивающего на печати строку.

При совместном использовании окружений figure и wrapfigure может случиться, что обтекаемая иллюстрация с меньшим номером печатается после необтекаемой иллюстрации с бо́льшим номером (и наоборот). В этом случае ничего не остается, как передвинуть одну из этих иллюстраций внутри исходного текста.

Окружение wrapfigure нельзя использовать непосредственно перед командой рубрикации, наподобие \section, а также внутри окружений, задающих перечни, и им подобных (itemize, enumerate, description, quote, quotation).

В стандартные поставки LATEX'а пакет wrapfig может и не входить; тогда его придется доставать из Интернета. См. приложение Ж по поводу того, где в Интернете хранятся TEX'овские материалы.

Дополнительные сведения об окружении wrapfigure можно получить из комментариев в файле wrapfig.sty, который, собственно говоря, и представляет собой стилевой пакет wrapfig.

В предшествующем тексте остались непроясненными два момента: какими командами задавать сам рисунок и как узнать его ширину? Ответ на второй вопрос зависит от ответа на первый. В следующей главе и в приложении Б будут даны два варианта ответов на эти вопросы.

8.3. Нештатные ситуации с плавающими объектами

Когда вы набираете исходный текст, заранее неясно, куда именно попадут плавающие иллюстрации (или таблицы; далее мы не будем всякий раз делать этой оговорки). Поэтому при просмотре и пробной печати возможны всяческие неожиданности.

Начнем с неприятности, подстерегающей вас при пользовании весьма привлекательным необязательным аргументом h («печатать прямо здесь!») у окружения figure или table. Если при этом, к несчастью, расположить иллюстрацию именно в указанном месте невозможно (потому что посредине иллюстрации должно быть место разрыва страницы), то LATEX действует так, словно в необязательном аргументе стояло не h, а ht. В результате иллюстрация будет напечатана вверху текущей или следующей страницы, а сообщение о происшедшем инциденте будет выдано на экран и в log-файл.

Далее, команда \suppressfloats запрещает печать любых плавающих иллюстраций (а также таблиц; мы не будем всякий раз это оговаривать) на той странице, на которую эта команда попала. Можно применить команду и с необязательным аргументом: если написать

```
\suppressfloats[t]
```

то вверху данной страницы иллюстрации размещаться не будут; если в качестве необязательного аргумента указать `b`, то иллюстрации заведомо не появятся внизу данной страницы.

С другой стороны, LaTeX предоставляет вам средство не затруднить, а, наоборот, облегчить размещение плавающих объектов (иллюстраций или таблиц). Именно, в необязательном аргументе окружения `figure` или `table` можно перед буквой `t`, `b` или `h` поставить восклицательный знак. В этом случае при размещении плавающего объекта LaTeX не будет обращать внимание на то, не слишком ли много иллюстраций попало на одну страницу и не слишком ли большую ее долю они займут (типичные причины, по которым LaTeX обычно перемещает плавающие иллюстрации вперед по тексту). У иллюстрации, начинающейся с команды

```
\begin{figure}[!t]
```

больше шансов быть напечатанной безотлагательно, чем в случае, если бы восклицательного знака не было.

Существуют и другие способы борьбы с причудами плавающих объектов. О них мы расскажем в разд. IX.7.

9. Еще о метках и ссылках

В разных местах этой книги уже шла речь о том, как можно помечать различные места документа, а затем на эти помеченные места ссылаться. В настоящем разделе мы систематизируем эту информацию. Дополнительные сведения о том, как можно влиять на вид ссылок, создаваемых командой `\ref`, читатель найдет в разд. VII.3.

9.1. Общие принципы

Как мы знаем, любое место в тексте можно пометить с помощью команды `\label` с одним аргументом; на помеченное место можно сослаться с помощью команды `\ref` или `\pageref` с тем же самым аргументом. Команда `\pageref` дает на печати номер страницы, на которую попала соответствующая метка; поэтому `\label` нужно ставить вплотную к тому слову, к которому относится ссылка (при наличии пробела слово и ссылка на него могут попасть на разные страницы).

Что же до команды `\ref`, то с ней дело обстоит так. Многие конструкции LaTeX'а автоматически нумеруют те или иные элементы документа. Из тех, с которыми мы уже сталкивались, можно назвать следующие:

- команды рубрикации текста (`\chapter`, `\section`, и т. п.); те из них (наиболее «мелкие»), что номеров разделов не печатают, влияния на команду `\ref` не оказывают;
- окружения, создающие нумерованные выключные формулы (такие, как `equation` и `eqnarray`, а также многочисленные окружения, определенные в пакете `amsmath`: `multline`, `gather`, `align` и иже с ними);
- команда `\caption`;
- команда `\item` в окружении `enumerate`;
- команда `\cite`.

Кроме того, автоматически создают номера, например, окружения типа «теорема», о которых пойдет речь в разд. VII.5.2; можно также самостоятельно создавать команды, дающие автоматическую нумерацию (см. гл. VII). Так или иначе, действует следующее правило:

> если в тексте стоит команда `\label{ghnm}`, то `\ref{ghnm}` выдает на печати последний из автоматически сгенерированных номеров, оказавшихся перед `\label{ghnm}`.

При первом (после появления новой команды `\label`) запуске LaTeX'а команды `\ref` и `\pageref` печатают вместо номеров вопросительные знаки, а на экран и в протокол трансляции выдается сообщение

```
LaTeX Warning: There were undefined references.
```

Если при дальнейшей работе над текстом номера, на которые ссылается `\ref` или `\pageref`, изменятся, LaTeX выдаст такое предупреждение:

```
LaTeX Warning: Label(s) may have changed.
   Rerun to get cross-references right.
```

Это означает, что в данный момент ссылки, сгенерированные командами `\ref` или `\pageref`, могут быть неверными. После повторного запуска LaTeX'а (иногда — не одного) все встает на свои места, и это предупреждение пропадает.

Скажем несколько слов про то, какие символы можно использовать в аргументе команды `\label`. Всегда можно пользоваться цифрами и (строчными и прописными) латинскими буквами; ни в коем случае нельзя помещать в аргумент `\label` фигурные скобки, а также симоволы `~` («тильда») или `\` («backslash»). Прочие символы в аргументе команды `\label` иногда безобидны, а иногда приводят к синтаксическим ошибкам. Пока вы не стали TEXником, лучше такие эксперименты не ставить.

Возможность использования русских букв в метках зависит от используемой вами русификации.

9.2. Визуализация меток

Возможность автоматической генерации ссылок, предоставляемая командами `\label` и `\ref`, — большое благо, но всякое техническое усовершенствовование приносит и новые проблемы. Предположим, что, рассматривая пробную распечатку, вы решили добавить ссылку на формулу 3.7. Писать прямо «3.7» рискованно: вдруг в процессе дальнейшей работы над текстом номер этой формулы изменится? Значит, надо воспользоваться командой `\ref`, но что писать в ее аргументе, если номер 3.7 присутствует только на печати, а в TEX'овском файле вместо этого номера при нужной вам формуле стоит команда `\label` с совершенно другим аргументом?

Можно, конечно, разыскать в файле формулу, соответствующую формуле 3.7 на печати, и скопировать аргумент команды `\label`, но есть и более удобный способ. Именно, если подключить стилевой пакет `showkeys`, то над каждым местом в тексте, помеченным с помощью команды `\label` (или, скажем, `\bibitem`), и над каждым местом, где стоит ссылка — команда `\ref` (или, скажем, `\cite`), будет надпечатываться и ваша метка — аргумент команды `\label`, `\ref` и т. п[8]. Иными словами, если ваша формула 3.7 в исходном тексте выглядела как

```
\begin{equation}
2\times 2=4,\label{main}
\end{equation}
```

то при просмотре и печати вы увидите над номером 3.7 надпись `main`, и сразу будет видно, как сослаться на эту формулу с помощью `\ref`.

Излишне объяснять, что перед беловой распечаткой строку

```
\usepackage{showkeys}
```

из преамбулы документа надо удалить.

9.3. Для любознательных

Скажем несколько слов о том, как происходит автоматическая генерация ссылок. Когда в обрабатываемом файле встречается команда `\label`, LATEX

[8]На верстку эти надпечатки не влияют.

записывает информацию о ней в специальный файл, называемый aux-файлом (при обработке файла `text.tex` имя aux-файла будет `text.aux`)[9]. При этом в aux-файл заносится следующая информация о метке: выбранное вами имя метки (аргумент команды `\label`), номер страницы, на которую эта метка попала (этот номер будет в дальнейшем напечатан командой `\pageref`), и, наконец, тот номер, который должен будет напечататься командой `\ref` (говоря более TEX'ническим языком, это вид на печати того счетчика, который последним подвергся увеличению с помощью `\refstepcounter` — см. гл. VII).

Далее, всякий aux-файл читается LATEX'ом за один сеанс работы дважды: первый раз до начала обработки текста и второй раз — после ее завершения. При первом чтении aux-файла LATEX запоминает имеющуюся в нем информацию о метках; именно исходя из этой информации команды `\ref` и `\pageref` печатают ссылки (если информации о данной метке при первом чтении aux-файла не обнаружено, вместо ссылки печатаются вопросительные знаки, а на экран выдается сообщение о неопределенной метке; так будет заведомо, если в tex-файле присутствуют ссылки на впервые появившуюся метку). При втором чтении aux-файла (после завершения работы с текстом, когда aux-файл был записан заново) LATEX сравнивает имеющуюся у него информацию о метках с той, что содержится в новой версии aux-файла; если информация о метках изменилась, выдается знакомое предупреждение «Label(s) may have changed».

10. Заметки на полях (маргиналии)

Заметки на полях страницы делаются с помощью команды `\marginpar` с единственным обязательным аргументом — текстом заметки. Если в исходном тексте написано

```
Маргиналии (фонарики) --- заголовки в виде
надписей\marginpar{!!!} на полях страниц.
```

то на печати вы увидите

!!! Маргиналии (фонарики) — заголовки в виде надписей на полях страниц.

Название `\marginpar` является сокращением английских слов, означающих «абзац на полях». Впрочем, текст заметки может состоять и из нескольких абзацев, разделяемых, как обычно, пустыми строками.

Если документ печатается в одну колонку и в «одностороннем» стиле (как в классах `article` или `report` без классовой опции `twoside`), то заметки выводятся по умолчанию на правое поле, а если документ печатается в одну колонку, но в «двустороннем» стиле, то на внешнее поле (правое, если страница имеет нечетный номер, и левое в противном

[9] Для каждого из файлов, включаемых в текст с помощью команды `\include`, создается отдельный aux-файл.

случае). Если документ печатается в две колонки, то заметка всегда выводится на ближайшее поле (ближайшее к той колонке, в которую попала заметка).

У команды `\marginpar` предусмотрен и необязательный аргумент. Он размещается *перед* обязательным; если эта команда использована с необязательным аргументом, то текст, выводящийся на поля, будет зависеть от того, на правое или на левое поле попадает заметка: на правое поле будет выведен текст, приведенный в обязательном аргументе, на левое — текст, приведенный в необязательном аргументе. Таким образом можно, например, вывести на поля стрелку, указывающую на текст:

```
\marginpar[$\Longrightarrow$]{$\Longleftarrow$}
```

(см. с. 48 по поводу команд, генерирующих стрелки в математических формулах).

По возможности заметки на полях помещаются на том же уровне, что и текст, к которым они относятся, но если этих заметок на каждой странице получается помногу (как в поэмах Кольриджа «Сказание о старом мореходе» или Маяковского «Про это»), то некоторые из них, во избежание наложений, будут сдвинуты вниз, а иногда даже перенесены на другую страницу (LaTeX сообщит об этом прискорбном событии во время трансляции).

Если текст набирается в одну колонку, то можно сделать так, чтобы заметки появлялись не на тех полях, на которых они должны быть согласно вышеописанным правилам, а на противоположных. Для этого надо дать команду `\reversemarginpar`. Существует еще и команда `\normalmarginpar`, возвращающая правила размещения заметок в исходное состояние.

Можно также менять параметры оформления самих заметок на полях. Эти параметры таковы:

`\marginparwidth`	ширина строки на полях;
`\marginparsep`	расстояние между полем и заметками;
`\marginparpush`	минимальное расстояние по вертикали между соседними заметками.

Значения этих параметров устанавливаются автоматически, в зависимости от класса документа. Вам может понадобиться их изменить, если вы меняете размер полей и/или ширину текста и при этом хотите пользоваться командой `\marginpar`.

Внутри «блоков» (например, внутри аргумента команды `\mbox` или внутри окружения `tabular`, предназначенного для верстки таблиц) команду `\marginpar` применять нельзя.

Глава V

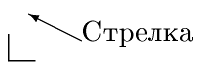
Псевдорисунки

Когда создавался TeX, а начиналось это в 1978 году, компьютерная графика была развита слабо. Поэтому операция по включению в текст рисунков в виде графических файлов в TeX'е не стандартизирована: TeX допускает импорт графического файла в текст с помощью команды `\special`, в аргументе которой содержится информация об импортируемом файле, но способ задания этой информации зависит от используемых `dvi`-драйверов, что снижает переносимость TeX'овских файлов. Чтобы как-то сгладить это неудобство, создатель LaTeX'а Лесли Лэмпорт предусмотрел возможность создания примитивных рисунков, состоящих из прямых и наклонных (с ограниченным репертуаром наклонов) линий, стрелок и окружностей (в принципе можно также рисовать кривые более или менее произвольной формы). В этой главе мы расскажем, как создавать такие «псевдорисунки».

Средства LaTeX'а, описываемые в этой главе, при профессиональной подготовке оригинал-макетов применяются редко: гораздо удобнее пользоваться PostScript-графикой. См. приложение Б.

1. Создание псевдорисунка и размещение на нем объектов

Псевдорисунки создаются с помощью окружения `picture`. Изучение этого окружения удобно начать с примера.




```
\begin{picture}(110,50)
\put(55,15){Стрелка}
\put(55,15){\vector(-2,1){40}}
\put(0,0){\line(1,0){20}}
\put(0,0){\line(0,1){20}}
\end{picture}
```

Разберем исходный текст, создавший этот рисунок: стрелку с надписью и уголок. На каждый псевдорисунок LaTeX должен отвести в тексте определенное место (после чего сам рисунок вполне может и выйти за пределы отведенного места: все зависит от того, что и где вы будете «рисовать»). Эти размеры задаются в *круглых* скобках через запятую немедленно после `\begin{picture}`, сначала ширина, затем высота (команды, связанные с псевдорисунками, — единственные в LaTeX'е, у которых в определенных случаях обязательный аргумент ставится не в фигурных скобках). Между скобками, запятой и числами, задающими размеры псевдорисунка, не должно быть пробелов (помните, что конец строки также воспринимается TeX'ом как пробел; если переноса на другую строку не избежать, воспользуйтесь знаком `%` для устранения получающегося пробела, как в примере на с. 18). По умолчанию ширина и высота псевдорисунка, и вообще все относящиеся к псевдорисункам размеры, задаются в пунктах (так и сделано в нашем примере). Можно указать любую единицу измерения размеров, относящихся к псевдорисункам: для этого надо изменить значение параметра `\unitlength` (см. с. 25 и далее по поводу параметров, являющихся длинами): если мы хотим, чтобы длины измерялись в миллиметрах, надо написать в преамбуле

```
\unitlength=1mm
```

(но не просто `mm`!). Размеры могут быть не только целыми, но и дробными числами, в которых нужно использовать десятичную точку (но не запятую).

Итак, место на псевдорисунок выделено. Чтобы поместить что-то на этот псевдорисунок, используется команда `\put` (внутри окружения `picture` писать текст «просто так» не следует). После `\put` в *круглых* скобках через запятую следуют координаты того объекта, который мы размещаем на псевдорисунке (сначала абсцисса, затем ордината; началом координат по умолчанию считается левый нижний угол псевдорисунка), а затем, без пробела, в фигурных скобках, — тот объект, который надо нанести. Для первой из наших команд `\put` этот объект был просто текстом, и соответственно в фигурных скобках только этот текст и был; для остальных трех команд, размещавших на рисунке стрелку и два отрезка, в фигурных скобках помещается нечто более сложное: описание этой стрелки и отрезков. В следующем разделе мы разберем, как такие описания устроены. Кстати, уголок в приведенном выше примере — не что иное, как левый нижний угол псевдорисунка (точка с координатами `(0,0)`).

Когда мы говорили о координатах объекта, имелись в виду координаты так называемой «точки отсчета» на этом объекте. Если объект —

текст, то точка отсчета — его левый нижний угол. Иногда при размещении текста удобней задать координаты его правого, а не левого нижнего угла. Чтобы так сделать, можно воспользоваться командой `\llap` с одним аргументом — текстом, чья точка отсчета будет в правом нижнем углу. В следующем примере точка отсчета «полужирной» кошки будет в левом нижнем углу, а «рубленой» — в правом нижнем.

Кошка**Кошка**

```
\begin{picture}(110,40)
\put(82,20){\textbf{Кошка}}
\put(80,20){\llap{\textsf{Кошка}}}
\end{picture}
```

Точка отсчета стрелки — ее начало. Когда пойдет речь о других объектах, размещаемых на псевдорисунке, мы будем указывать, где расположены их точки отсчета.

Еще несколько общих правил, относящихся к окружению `picture`. Во-первых, внутри этого окружения не должно быть пустых строк. Во-вторых, необходимо сказать о том, как окружение `picture` взаимодействует с окружающим текстом. Весь псевдорисунок, порождаемый этим окружением, рассматривается TEX'ом как одна большая буква, ширина и высота которой заданы в скобках через запятую после `\begin{picture}`, так что если окружение `picture` встретилось в середине абзаца, эта «буква» будет помещена в строку, причем соседние строки раздвинутся, чтобы она поместилась. Если это не то, чего вы хотите, — начинайте окружение `picture` между абзацами (после пустой строки или команды `\par`). Можно также поместить окружение `picture` внутри окружения наподобие `flushright` или `center` — при этом LATEX автоматически установит разумные интервалы между псевдорисунком и окружающим текстом. Совершенно безбоязненно можно помещать окружение `picture` внутри «плавающего» окружения `figure` или `table` (см. разд. IV.8).

Если вы оформляете псевдорисунок как рисунок в оборку с помощью окружения `wrapfigure` (при подключенном пакете `wrapfig` — см. разд. IV.8.2), то ширину рисунка, которую, как вы помните, необходимо задать во втором обязательном аргументе окружения `wrapfigure`, можно задать как кратное параметра `\unitlength`. Например, если псевдорисунок начинается с

```
\begin{picture}(50,43)
```

то окружение `wrapfigure` надо начать как-нибудь так:

```
\begin{wrapfigure}{o}{50\unitlength}
```

Кроме текста, на псевдорисунках можно размещать отрезки, стрелки, окружности, круги и овалы (прямоугольники с закругленными углами). Далее мы опишем, как задавать эти объекты.

2. Отрезки и стрелки

Отрезки задаются с помощью команды `\line`. LaTeX'у надо сообщить наклон и размер отрезка. Вот пример команды `\put`, выводящей отрезок:

```
\begin{picture}(100,50)
\put(60,50){\line(1,-2){20}}
\end{picture}
```

Как мы уже понимаем, здесь на рисунок размера 100×50 пунктов наносится отрезок с началом в точке $(60, 50)$. Наклон отрезка задается парой целых чисел, расположенных в *круглых* скобках через запятую непосредственно после `\line`. Отношение этих чисел должно быть равно «угловому коэффициенту» отрезка (тангенсу угла наклона к горизонтали); в нашем случае эти числа суть `(1,-2)`, это означает, что отрезок отклоняется «на одну единицу вправо и на две единицы вниз». Если эти числа `(1,0)`, то отрезок горизонтален, если `(0,1)`, то отрезок вертикален.

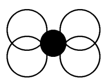
Размер отрезка задается в фигурных скобках после круглых скобок, в которых задан наклон. Этот размер, вообще говоря, — не его длина, но длина его проекции на горизонтальную ось (кроме случаев, когда отрезок вертикален — тогда задается его длина по вертикали).

Длину отрезка можно (если она не слишком мала) задавать произвольно, а вот наклон — нет. Каждое из целых чисел, задающих наклон, не должно превосходить 6 по абсолютной величине, и, кроме того, эти два числа не должны иметь общих делителей, кроме 1 (это последнее условие репертуар возможных наклонов не ограничивает).

Стрелки задаются с помощью команды `\vector`, которая нам уже встречалась в примере. Синтаксис этой команды совершенно такой же, как у команды `\line`: в круглых скобках пишется пара чисел, задающая наклон стрелки, а затем в фигурных скобках параметр, задающий ее размер (длина проекции на горизонтальную ось, если стрелка не вертикальна, и длина проекции на вертикальную ось, если стрелка вертикальна). Отличие от команды `\line` в том, что репертуар возможных наклонов стрелок еще более ограничен, чем у отрезков: целые числа, задающие наклон, не должны превосходить 4 по абсолютной величине (и по-прежнему не должны иметь общих делителей). Точкой отсчета стрелки является ее начало.

3. Окружности, круги и овалы

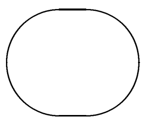
Окружность задается командой `\circle`, а круг (сплошной черный кружок) — ее вариантом «со звездочкой» `\circle*`. У этих команд единственный аргумент — диаметр круга или окружности. Как обычно, он задается в единицах, равных значению параметра `\unitlength` (по умолчанию — в пунктах). Точкой отсчета окружности или круга является центр. Вот пример картинки с окружностями и кругами:



```
\begin{picture}(100,80)
\put(30,30){\circle{30}}
\put(70,30){\circle{30}}
\put(30,50){\circle{30}}
\put(70,50){\circle{30}}
\put(50,40){\circle*{20}}
\end{picture}
```

Количество реально возможных диаметров кругов ограничено. Если окружности или круга с диаметром, указанным в качестве аргумента команды `\circle` или `\circle*`, в LaTeX'овских шрифтах нет, то будет напечатана окружность (круг), диаметр которой наиболее близок к указанному.

Наряду с окружностями и кругами, на псевдорисунок можно нанести также «овал» — прямоугольник с закругленными углами. Он задается командой `\oval`, аргументы которой — ширина и высота овала. Эти аргументы задаются в *круглых* скобках через запятую. Точка отсчета овала — его центр. Пример:

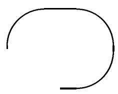


```
\begin{picture}(100,80)
\put(50,40){\oval(100,80)}
\end{picture}
```

Кроме того, возможны и «неполные» овалы, представляющие собой половины или четверти от полных. Чтобы задать такой неполный овал, надо задать команде `\oval` необязательный аргумент (в квадратных скобках, после обязательного). Для задания половины овала этот аргумент должен быть одной из следующих букв:

`t`	верхняя половина;
`b`	нижняя половина;
`r`	правая половина;
`l`	левая половина.

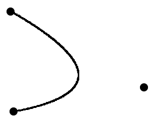
Для задания четверти овала необязательный аргумент команды `\oval` должен быть сочетанием двух из этих букв (например, `tr` для верхней правой четверти). Точка отсчета усеченного овала расположена там же, где точка отсчета соответствующего ему полного овала. Вот пример картинки с усеченными овалами:



```
\begin{picture}(100,80)
\put(50,40){\oval(80,60)[t]}
\put(50,40){\oval(80,60)[br]}
\end{picture}
```

4. Кривые

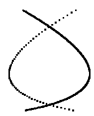
При пользовании окружением `picture` вы имеете возможность нанести на псевдорисунок кривую более или менее произвольной формы (эти кривые — так называемые квадратичные сплайны Безье). Это делается с помощью команды `\qbezier`. Вот пример ее работы:



```
\begin{picture}(80,80)
\qbezier(22,2)(120,20)(20,77)
\put(22,2){\circle*{5}}
\put(120,20){\circle*{5}}
\put(20,77){\circle*{5}}
\end{picture}
```

После `\qbezier` надо указать (без пробелов) координаты трех точек: начальной, «опорной» и конечной. Из начальной точки кривая выходит, устремляется к опорной, но, как правило, до нее не доходит, поскольку сворачивает к конечной точке, в которой и заканчивает свой путь. В нашем примере мы для ориентировки нанесли на псевдорисунок черные кружки в этих трех точках.

Никакой мистики в том, что TEX рисует кривые, нет: эти кривые просто составляются из сотен черных квадратиков. Можно попросить LATEX не так густо ставить квадратики, из которых состоит кривая. Для этих целей у команды `\qbezier` предусмотрен необязательный аргумент — количество этих квадратиков. Он ставится *перед* всеми обязательными в квадратных скобках:



```
\begin{picture}(80,80)
\qbezier(22,2)(120,20)(20,77)
\qbezier[60](58,2)(-40,20)(60,77)
\end{picture}
```

Кстати, обратите внимание, что опорная точка второй из наших кривых находится где-то за пределами текста. Это не страшно, поскольку ее координаты используются LATEX'ом только для расчетов.

Какой бы необязательный аргумент команды `\qbezier` мы ни задавали, количество квадратиков, из которых составляется кривая, не превысит числа 500. Если вы решили увеличить этот максимум, допустим, до тысячи, надо написать так:

```
\renewcommand{\qbeziermax}{1000}
```

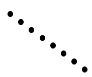
Если так вы напишете в преамбуле, то предел 1000 будет относиться ко всем кривым в вашем тексте, а если внутри группы (например, внутри окружения), то изменение этого параметра забудется по выходе из группы. В гл. VII будет объяснено, что означает `\renewcommand` в общем случае.

5. Дополнительные возможности

Иногда бывает нужно нанести на псевдорисунок несколько регулярно расположенных объектов. В этом случае, вместо того чтобы много раз писать `\put`, удобно воспользоваться командой `\multiput`. Она располагает на псевдорисунке несколько одинаковых объектов на равных расстояниях. Синтаксис этой команды таков:

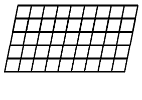
`\multiput(`x`,`y`)(`Δx`,`Δy`){`n`}{объект}`

Здесь x и y — координаты первого из размещаемых объектов (как и в обычной команде `\put`), Δx и Δy — расстояния, на которые каждый следующий объект будет сдвинут относительно предыдущего по горизонтали и вертикали, n — количество объектов, которые надо разместить, и, наконец, *объект* — это, как и у команды `\put`, описание того, что мы размещаем на рисунке. Пример:



```
\begin{picture}(100,80)
\multiput(10,70)(8,-6){8}%
{\circle*{3}}
\end{picture}
```

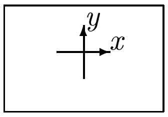
Обратите внимание на использование знака процента для удаления нежелательного пробела, создаваемого концом строки. Вот еще один пример; здесь с помощью команды `\multiput` рисуется решеточка:



```
\begin{picture}(100,50)
\multiput(0,0)(10,0){10}%
{\line(1,5){10}}
\multiput(0,0)(2,10){6}%
{\line(1,0){90}}
\end{picture}
```

Использование команды \multiput так, как это было сделано в предыдущем примере, ведет к неоправданным затратам машинного времени. Например, каждый из наклонных отрезков собирается из маленьких символов, причем TEX'у приходится повторять эту скучную операцию 10 раз. Разумнее было бы собрать этот отрезок лишь единожды, а дальше его просто копировать. Это можно сделать с помощью «блоковых переменных». Мы расскажем об этом в гл. VIII.

Иногда, когда псевдорисунок достаточно сложен, удобно применить следующий прием: задать в качестве аргумента одной из команд \put целое окружение picture (точкой отсчета будет служить левый нижний угол). При этом вы сможете отсчитывать координаты объектов на «подрисунке» относительно самогó подрисунка, а не внешнего рисунка, что часто бывает проще; кроме того, если понадобится сдвинуть этот «подрисунок» как единое целое, то для этого будет достаточно изменить аргументы в одной-единственной команде \put. Вот пример рисунка с подрисунком (будем считать, что это классная доска, на которой нарисованы оси координат):



Этому рисунку соответствовал такой исходный текст:

```
\begin{picture}(120,80)
% Края доски:
\put(0,0){\line(1,0){120}} \put(0,80){\line(1,0){120}}
\put(0,0){\line(0,1){80}} \put(120,0){\line(0,1){80}}
% Оси координат:
\put(40,25){\begin{picture}(40,40)%
             \put(20,0){\vector(0,1){40}}
             \put(0,20){\vector(1,0){40}}
             \put(40,22){$x$} \put(22,40){$y$}
           \end{picture}}
\end{picture}
```

Кстати говоря, размеры внутренней картинки можно было бы задать совершенно произвольно, например, (200,200) или даже (0,0) — команда \put бездумно размещает объекты таким образом, чтобы их точки отсчета имели указанные координаты, и при этом не интересуется, сколько места они реально занимают и не наложатся ли на текст или другие объекты.

Нередко требуется сдвигать не какую-то часть псевдорисунка, а весь псевдорисунок как целое (например, если вы ищете оптимальное расположение иллюстрации по отношению к тексту). Для этого удобно использовать еще одну возможность окружения `picture`: можно задать его таким образом, чтобы начало координат было не в левом нижнем углу, а в любой другой точке. Для этого после `\begin{picture}` надо задать не одну, а две пары чисел в круглых скобках. В этом случае первая пара чисел будет, как и прежде, обозначать ширину и высоту места, выделяемого LaTeX'ом на псевдорисунок, а вторая пара будет указывать, каковы координаты левого нижнего угла этого псевдорисунка (по умолчанию, т. е. если второй пары чисел в круглых скобках нет, они были бы просто `(0,0)`). Главное только — не напутать со знаками: если вы сказали

`\begin{picture}(`a,b`)(`x,y`)`

то это значит, что левый нижний угол псевдорисунка имеет координаты (x, y), стало быть, по сравнению со случаем, когда $x = y = 0$, весь псевдорисунок сдвинется на $-x$ по горизонтали и на $-y$ по вертикали! Если вы ничего не поняли, посмотрите на следующий пример, в котором второй псевдорисунок сдвигается на 20 единиц вправо и на 10 единиц вверх по отношению к первому:


```
\begin{picture}(150,80)
\put(0,0){\line(1,0){140}}
\put(0,70){\line(1,0){140}}
\put(0,0){\line(0,1){70}}
\put(140,0){\line(0,1){70}}
\put(25,30){\Huge Сдвиг}
\end{picture}\\[25pt]
\begin{picture}(150,80)(-20,-10)
\put(0,0){\line(1,0){140}}
\put(0,70){\line(1,0){140}}
\put(0,0){\line(0,1){70}}
\put(140,0){\line(0,1){70}}
\put(25,30){\Huge Сдвиг}
\end{picture}
```

6. Параметры оформления псевдорисунка

Про один из таких параметров мы уже говорили — это единица измерения длин на псевдорисунке, обозначаемая `\unitlength`.

В какой-то мере можно регулировать и толщину линий на наших псевдорисунках. Для этого предусмотрены команды `\thinlines` (тонкие линии) и `\thicklines` (толстые линии). По умолчанию стоит режим, в котором линии будут тонкими. Команды `\thicklines` и `\thinlines` можно давать не только в преамбуле, но и в самом тексте (в том числе и внутри окружения `picture`, так что можно регулировать, какие линии будут толстыми, а какие тонкими). Если одна из этих команд дана внутри группы, то по окончании группы ее действие прекращается (не забывайте, что любое окружение само по себе образует группу).

Кроме того, можно задать произвольным образом толщину вертикальных и горизонтальных (но не наклонных!) линий. Для этих целей служит команда `\linethickness`. У этой команды один обязательный аргумент — толщина линий, выраженная в TeX'овских единицах длины. Если мы скажем

```
\linethickness{2.5mm}
```

то все вертикальные и горизонтальные отрезки на псевдорисунке будут иметь толщину 2.5 мм, и такой же будет сторона квадратиков, из которых составляются кривые.

Глава VI

Печать текста с выравниванием

При работе на пишущей машинке печать таблиц, состоящих из нескольких колонок, не вызывала особых проблем: все литеры имели одинаковую ширину. Однако полиграфические шрифты (в частности, используемые TEX'ом) являются, как правило, «пропорциональными» (каждая буква имеет свою ширину), и в этом случае добиться выравнивания в колонках сложнее.

В настоящей главе мы рассмотрим два основных способа, предоставляемых LATEX'ом для печати текста с выровненными колонками (например, таблиц). Начнем с менее мощного, но более простого — имитации табулятора.

1. Имитация табулятора

1.1. Элементарные средства

Табулятор имитируется в LATEX'е с помощью окружения `tabbing`. При печати таблиц с помощью этого окружения пользователь сам задает места, в которых должна начаться очередная колонка. Конкретно это выглядит так. При наборе первой строки этого окружения можно в любой момент поставить команду `\=` — она отмечает очередное место, с которого начинается новая колонка («позицию табуляции»). Это место (расстояние от начала строки) запоминается, и в дальнейшем можно с помощью команды `\>` «перескочить» к очередной позиции табуляции — текст, следующий после этой команды, будет набираться, начиная с позиции табуляции. Строки разделяются командой `\\`. Рассмотрим это на примере:

начало	середина	конец
раз	два	три
раз	два	три
начинаем	продолжаем	заканчиваем

```
\begin{tabbing}
начало\quad\=середина
\quad\=конец\\
раз\>два\>три\\
 раз\> два\> три\\
начинаем\>
продолжаем\>
заканчиваем\\
\end{tabbing}
```

В первой строке мы задали две позиции табуляции двумя командами `\=` (на всякий случай мы разделили дополнительными пробелами слова в первой строке и, тем самым, наши позиции табуляции; отсюда команды `\quad`). Первая строка завершается командой `\\`, а во второй строке мы начинаем установленными позициями табуляции пользоваться. Слово «раз» напечаталось с начала строки (каждая строка начинается с крайней левой позиции, если отсутствует команда вроде `\>`, задающая переход к новой позиции). Далее идет команда `\>` — «перейти на следующую позицию табуляции». И действительно, следующее после нее слово «два» начинается со второй позиции — как раз там же, где начиналось слово «середина». Перед словом «три» стоит еще одна команда `\>` — оно печатается с третьей позиции, как раз под словом «конец», с начала которого мы эту позицию и определили. Третья строка ничем не отличается от четвертой, хотя в исходном тексте между командами `\>` и словами стоят пробелы. Дело в том, что *пробелы после команд* `\>` *игнорируются.* Наконец, в четвертой строке слова при печати наложились друг на друга. Это и не удивительно: окружение `tabbing` исправно начинает очередную порцию текста с той позиции, которую мы ему указали, но при этом не проверяет, сколько места этот текст реально займет и не будут ли перекрываться колонки — за это целиком отвечает тот, кто текст готовит. Видимо, в данном случае следовало оставить побольше места при определении позиций табуляции (например, написать в первой строке `\qquad` вместо `\quad`).

Кроме установки дополнительных интервалов экспериментальным путем, есть и другой способ правильно проставить позиции табуляции. Именно, если закончить строку не командой `\\`, а командой с суровым названием `\kill`, то эта строка не будет напечатана, но все позиции табуляции, установленные в ней, будут запомнены LaTeX'ом, и их можно будет использовать в последующих строках. В приведенном выше примере можно было бы написать так:

начало	середина	конец
раз	*два*	три
начинаем	продолжаем	заканчиваем

```
\begin{tabbing}
начинаем \=продолжаем \=
заканчиваем\kill
начало\>середина\>конец\\
\bfseries раз\>
\itshape два\>три\\
начинаем\>продолжаем\>
заканчиваем\\
\end{tabbing}
```

Обратите внимание, что при установке позиций табуляции в первой (не печатающейся) строке мы сделали пробелы между концом слова и командой `\=` (иначе в последней строке слова бы опять слились: нам нужно, чтобы первая позиция табуляции не была впритык к концу слова «начинаем»). Заметьте также, что во второй строке мы убрали команды `\quad`; можно было бы их и оставить — на внешний вид таблицы это бы никак не повлияло, поскольку позиции табуляции уже установлены и лишние пробелы перед очередной командой `\>` никого не волнуют. По этой же причине мы не потрудились оставить пробелы между словами и `\>` в строке «начинаем, продолжаем, заканчиваем». Наконец, обратите внимание и на то, как мы меняли шрифт в строке «раз, два, три»: слово «три» переключилось на обычный шрифт само собой. Это объясняется тем, что *часть текста окружения* `tabbing`, *расположенная между двумя командами* `\>` *или* `\=`, *образует группу.*

Внутри окружения `tabbing` используется команда `\=`, которая, как мог заметить читатель, обычно имеет совсем другой смысл — постановка диакритического знака над буквой (см. таблицу на с. 106). Команды `\'` и `` \` `` также имеют внутри этого окружения особый смысл, о котором пойдет речь ниже. Поэтому, если внутри `tabbing` нам понадобился диакритический знак (скажем, над буквой e), то надо руководствоваться такой таблицей:

Внутри окружения `tabbing`

вместо	надо набирать
`\=e`	`\a=e`
`\'e`	`\a'e`
`` \`e ``	`` \a`e ``

1.2. Более сложные средства

Интервалы и разрывы между строками. Команда `\\` внутри окружения `tabbing` может иметь необязательный аргумент, действующий формально так же, как для этой команды, употребляемой внутри абзаца: если в квадратных скобках поставить длину (измеренную в воспринимаемых TEX'ом единицах, см. разд. I.2.10, или же какой-либо LATEX'овский параметр, значением которого является длина, например, `\medskipamount`), то после этой строки будет сделан дополнительный интервал, величина которого равна указанной длине. Имеет команда `\\` и «вариант со звездочкой»: если написать `\\*` вместо `\\`, то после строки, завершаемой этой командой, начинать новую страницу будет запрещено. Команда `\\*` также может принимать необязательный аргумент. Он имеет тот же смысл, что и для соответствующей команды без звездочки.

Переустановка позиций табуляции. Команды `\=`, устанавливающие позиции табуляции, можно давать не только в первой строке. Сначала пример:

парочка	позиций		табуляции	
	плюс		еще одна здесь:	
теперь	их		уже	три
Вторую	мы	сменим		и посмотрим:
где	эти	позиции		теперь

```
\begin{tabbing}
парочка \=позиций
\=табуляции\\
\>плюс\>еще
одна здесь:\=\\
теперь\>их\>
уже\>три\\
Вторую \>мы\quad
\=сменим \>
и посмотрим:\\
где\>эти\>
позиции\>теперь\\
\end{tabbing}
```

Опишем точно, как команда `\=` взаимодействует с `\>`. Внутри окружения `tabbing` в каждый момент LATEX'у известно некоторое количество позиций табуляции, занумерованных подряд, от нуля до какого-то целого числа (не более двенадцати). При входе в окружение известна только позиция с номером нуль (это всегда начало строки). Увеличиваться число известных позиций может за счет команды `\=`, используются позиции табуляции командой `\>`. Если команда `\=` встречается в строке *после* того, как использованы все известные позиции табуляции, то количество известных позиций табуляции увеличивается на 1 и очередная позиция табуляции устанавливается в месте, куда попала команда `\=`. Если же `\=` встречается в строке *до* того, как все известные позиции табуляции израсходованы, то новых известных позиций не прибавляется, просто очередная по счету позиция табуляции заменяется на ту, которую задает команда `\=`.

Иногда бывает необходимо в пределах одной и той же таблицы временно перейти на новое расположение позиций табуляции, а затем вернуться к прежнему. Для этого используются команды `\pushtabs` и `\poptabs`. Первая из них запоминает расположение позиций табуляции; после этой команды можно позиции переустановить, пользоваться этими новыми переустановленными позициями... — после команды `\poptabs` значения старых позиций табуляции будут восстановлены. Пример:

раз	два	три	четыре
гиппопотам		аллигатор	
раз		два	
три		четыре	
one	two	three	four
viens	divi	trīs	četri

```
\begin{tabbing}
раз\quad\=два\quad\=три\quad\=четыре\\
\pushtabs гиппопотам\quad\=аллигатор\\
раз\>два\\ три\>четыре\\
\poptabs
one\>two\>three\>four\\
viens\>divi\>tr\a=\i s\>\v{c}etri\\
\end{tabbing}
```

Команды `\pushtabs` и `\poptabs` должны быть «сбалансированы»: каждой команде `\pushtabs`, запоминающей позиции табуляции, должна соответствовать вспоминающая их команда `\poptabs`. Если это условие не выполнено, вы получите сообщение об ошибке. Обратите также внимание, что знак долготы над буквой i в слове trīs («три» по-латышски) мы поставили с помощью команды `\a`.

Экзотика. Для полноты картины опишем некоторые изысканные возможности окружения `tabbing`.

Команда `\'` (внутри окружения `tabbing`) размещает текст таким образом, чтобы он не начинался, а *заканчивался* у позиции табуляции. Сама эта команда позиций табуляции «не тратит»; просто весь текст, размещенный между `\>` или `\=` и `\'`, размещается левее позиции табуляции, определяемой командой `\>` или `\=`. Таким способом можно верстать таблицы, в которых колонки выровнены по правому краю, а не по левому, как получается при обычном использовании `tabbing`. Вот пример:

слева	справа
à gauche	à droite
links	rechts
pa kreisi	pa labi

```
\begin{tabbing}
\hspace{3.5cm}\=\kill
слева\>справа\'\\
\a`a gauche\>\a`a droite\'\\
links\>rechts\'\\
pa kreisi\>pa labi\'
\end{tabbing}
```

Еще раз обратите внимание, что для постановки диакритического знака над буквой а нам пришлось писать `\a‘` вместо `\‘` (см. с. 208).

Команда `\‘` внутри окружения `tabbing` прижимает весь текст строки, идущий после нее, к правому краю; между этой командой и командой, завершающей строку, не должно быть команд, использующих или устанавливающих позиции табуляции. Например, таблицу, у которой первая колонка выровнена по левому краю, а вторая — по правому (как в предыдущем примере), можно было бы задать так:

слева	справа
à gauche	à droite
links	rechts
pa kreisi	pa labi

```
\begin{tabbing}
слева\‘справа\\
\a‘a gauche\‘\a‘a droite\\
links\‘rechts\\
pa kreisi\‘pa labi\\
\end{tabbing}
```

Кстати, здесь нам вообще не понадобилось устанавливать позиции табуляции. Впрочем, смотрится эта таблица неважно.

Как мы уже отмечали, при начале новой строки текст начинается с нулевой позиции табуляции, т. е. с начала строки. Команда `\+` позволяет изменить такое положение вещей: после этой команды при начале каждой новой строки текст будет начинаться не с нулевой, а с первой позиции табуляции (как если бы каждая последующая строка начиналась с команды `\>`). Если дать еще одну команду `\+`, то текст в последующих строках будет начинаться уже и не с первой, а со второй позиции, и т. д. Команда `\-` внутри окружения `tabbing` означает вовсе не место, где можно перенести слово (впрочем, команда с таким действием в этом окружении и не нужна): она действует противоположно команде `\+`. Наконец, команда `\<`, будучи употребленной в начале строки (в других местах ее употреблять нельзя), действует аналогично `\-`, но в пределах только этой строки (а не всех последующих, как `\+` и `\-`). Следующий пример иллюстрирует все эти изыски:

раз	два	три	четыре
	два		
		три	
			четыре
		три	
			четыре
	два		
раз	два	три	четыре

```
\begin{tabbing}
раз \=два \=три \=\kill
раз\>два\>три\>четыре\+\\
два\+\\ три\+\\ четыре\\
\<три\\ четыре\-\-\\
два\-\\
раз\>два\>три\>четыре\\
\end{tabbing}
```

Описанные в этом разделе возможности окружения `tabbing` на практике используются редко, поскольку для печати сложных таблиц в LaTeX'е есть более удобное средство — окружение `tabular`. Перейдем к его описанию.

2. Таблицы

При пользовании окружением `tabbing` вы должны самостоятельно следить, чтобы разные колонки не накладывались друг на друга. Можно, однако, передать эти заботы программе: TeX предоставляет возможности для печати таблиц, в которых ширина колонок выбирается автоматически (по максимальной ширине их содержимого). В LaTeX'е для этих целей используются окружения `tabular` (для набора таблиц с текстом) и `array` (для набора таблиц из формул). Помимо автоматизированного определения ширины колонки, эти окружения дают возможность печатать разлинованные таблицы, таблицы, в которых некоторые записи охватывают несколько колонок, и т. д. В гл. II уже шла речь про окружение `array`; здесь мы подробно разберем, как работает `tabular`; все возможности этого окружения, о которых идет речь в этой главе, доступны и для `array`, и ниже мы дадим примеры их использования.

2.1. Простейшие случаи

Окружение `tabular` задает таблицу. Окружению необходимо задать обязательный аргумент — *преамбулу таблицы*. Преамбула, помещаемая в фигурных скобках непосредственно после `\begin{tabular}`, представляет собой, в простейшем случае, последовательность букв, описывающих структуру колонок таблицы (по букве на колонку). Буквы эти могут быть такими:

`l` означает колонку, выровненную по левому краю;

`r` означает колонку, выровненную по правому краю;

`c` означает колонку с центрированным текстом.

Между `\begin{tabular}` (с преамбулой) и закрывающей окружение командой `\end{tabular}` располагается собственно текст таблицы. В нем команда `\\` разделяет строки таблицы, а знак `&`, называемый «амперсендом», разделяет колонки таблицы внутри одной строки (так что текст между двумя ближайшими амперсендами описывает «одну графу» таблицы). Пробелы в начале или конце «графы» таблицы игнорируются. Если вы прочли мелкий шрифт в разд. II.4.2, то могли заметить буквальное совпадение с тем, что там написано про окружение `array`.

Разница лишь в том, что содержимое граф таблицы обрабатывается в окружении `tabular` как текст, а в окружении `array` — как формулы. Вот первый пример:

Тип перечня	нумерация
`itemize`	нет
`enumerate`	есть
`description`	нет

```
\begin{tabular}{lc}
Тип перечня & нумерация \\[5pt]
\ttfamily itemize & нет\\
\ttfamily enumerate & есть\\
\ttfamily description & нет\\
\end{tabular}
```

Обратите внимание на две вещи. Во-первых, команда `\\`, завершающая первую строку, дана с необязательным аргументом. Он задается так же и имеет тот же смысл, как если бы эта команда была внутри абзаца (с. 122) или окружения `tabbing` (с. 209): после строки вставляется дополнительный вертикальный промежуток (кстати, между строками таблицы, определенной с помощью окружения `tabular`, разрыва страницы *никогда* не происходит, так что в этом окружении у команды `\\` варианта «со звездочкой» нет). Во-вторых, команда `\ttfamily` всякий раз меняла шрифт только в одной графе таблицы, не действуя на соседние. Это объясняется тем, что *графа таблицы образует группу,* так что любые изменения параметров (в том числе текущего шрифта), проведенные в одной графе, не влияют на остальные.

Прежде чем мы начнем говорить о более сложных вещах, скажем о том, как окружение `tabular` взаимодействует с текстом вне его. Подобно окружению `picture`, оно не начинает печать с новой строки и не завершает текущего абзаца. Вся таблица, порождаемая этим окружением, рассматривается TEX'ом как одна большая буква; если окружение `tabular` встретилось в середине абзаца, эта «буква» будет помещена в строку (соседние строки раздвинутся, чтобы она поместилась), и результат будет выглядеть некрасиво. Если такое размещение текста не входит в ваши планы, начинайте окружение `tabular` между абзацами (после пустой строки или команды `\par`). Удобно также бывает поместить окружение `tabular` внутрь окружения `center` или подобного ему: тогда LATEX сам позаботится о пробелах между таблицей и окружающим текстом.

Иногда бывает полезно знать, как расположена «большая буква», представляющая собой окружение `tabular`, по отношению к строке, в которой она оказалась. Ответ: ее середина идет вровень с низом строки (точнее, с «базисной линией» — см. гл. VIII); соответственно, на половинной высоте находится и точка отсчета этой «буквы». Пример:

слово

А	Б
В	Г

слово

```
слово
\begin{tabular}{rr}
А & Б\\В & Г
\end{tabular}
слово
```

Можно также использовать окружение `tabular` с необязательным аргументом `b`: тогда «буква», созданная окружением `tabular`, будет выровнена по нижней строке; необязательный аргумент `t` дает выравнивание по верхней строке:

слово

А	Б
В	Г

А	Б
В	Г

А	Б
В	Г

```
слово
\begin{tabular}{rr}
А & Б\\В & Г
\end{tabular}
\begin{tabular}[t]{rr}
А & Б\\В & Г
\end{tabular}
\begin{tabular}[b]{rr}
А & Б\\В & Г
\end{tabular}
```

Как мог заметить читатель, необязательный аргумент в данном случае ставится перед обязательным.

Можно напечатать и разлинованную таблицу. Для этого применяются команды, создающие горизонтальные и вертикальные отрезки («линейки» на полиграфическом жаргоне — см. разд. III.10). Горизонтальные отрезки задаются с помощью команды `\hline`. Эта команда может следовать либо непосредственно после `\\` (тогда отрезок печатается после строки, завершенной этим `\\`), либо непосредственно после `\begin{tabular}` и преамбулы (тогда отрезок печатается перед началом таблицы). Задаваемый командой `\hline` горизонтальный отрезок имеет ширину, равную общей ширине таблицы. Что касается вертикальных отрезков, то давайте для начала также ограничимся случаем, когда эти отрезки, разделяющие колонки таблицы, простираются на всю ее высоту, сверху донизу. Такие отрезки проще всего предусмотреть в преамбуле таблицы. До сих пор мы говорили, что преамбула таблицы — это последовательность из букв `l`, `c` или `r`, характеризующих колонки. На самом деле в преамбуле может присутствовать и информация, описывающая то, что должно быть между колонками таблицы. В частности, символ `|`, помещенный в преамбулу таблицы между буквами, описывающими колонки, задает вертикальную линейку, разделяющую эти колонки. Можно поставить символ `|` перед первой из этих букв или после последней — тогда вертикальная линейка будет ограничивать таблицу слева или справа. Несколько таких символов могут стоять подряд — тогда колонки будут разделяться не одинарной, а двойной (тройной и т. д.) вертикальной линейкой. Вот пример разлинованной таблицы:

слон	zilonis
бегемот	nilzirgs
лев	lauva

```
\begin{tabular}{||l|l||}
\hline
слон & zilonis\\
бегемот & nilzirgs\\
лев & lauva\\
\hline
\end{tabular}
```

Две команды `\hline` могут следовать одна непосредственно за другой; в этом случае на печати получатся две горизонтальные линейки, одна под другой, разделенные по вертикали небольшим интервалом. Если слева и справа таблица ограничена вертикальными линейками, то на пересечении крайних вертикальных линеек с горизонтальными на печати получится разрыв:

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

```
\begin{tabular}{|c|c|}
\hline
Северо-Запад & Северо-Восток\\
\hline
\hline
Юго-Запад & Юго-Восток\\
\hline
\end{tabular}
```

Позже мы расскажем, как от этого разрыва можно избавиться.

2.2. Более сложные случаи

Надписи, охватывающие несколько колонок. Чтобы создать такую надпись, нужно на месте соответствующей графы таблицы записать команду `\multicolumn`. У этой команды три обязательных аргумента:

1) Количество колонок, охватываемых нашей «нестандартной» графой.

2) «Преамбула» нашей графы. В качестве таковой может выступать буква `l`, `r` или `c` (текст в графе был прижат влево, вправо или центрирован), возможно, с символами `|` слева или справа, если мы хотим, чтобы графа была ограничена вертикальными линейками.

3) Текст, записываемый в графу.

Таблица VI.1.

Популярные напитки			
Название	Старый мельник	Бочкарев	Очаковское
Атрибут	Душевное	Правильное	Живительное
Цена	15		14

Пример — в таблице VI.1.

Получается эта таблица таким образом (обратите внимание, что в строке с ценами в командах `\multicolumn` вертикальная черта стоит справа от `c`, но не стоит слева — почему так надо, объяснено на с. 217):

```
\begin{tabular}{|l|l|l|l|}
\hline
\multicolumn{4}{|c|}{\textbf{Популярные напитки}}\\
\hline
Название & Старый мельник & Бочкарев & Очаковское\\
\hline
Атрибут&Душевное & Правильное & Живительное\\
\hline
Цена & \multicolumn{2}{c|}{15} & \multicolumn{1}{c|}{14}\\
\hline
\end{tabular}
```

Если таблица, в которой вы используете `\multicolumn`, является к тому же еще и линованной, то возможностей команды `\hline` для рисования горизонтальных отрезков может не хватить: иногда бывает нужен горизонтальный отрезок, простирающийся не на всю ширину таблицы, а охватывающий только часть ее колонок. Для рисования таких отрезков предусмотрена команда `\cline`. Как и `\hline`, ее нужно давать сразу после `\\` , но она имеет обязательный аргумент — номера первой и последней из колонок, охватываемых горизонтальной чертой, разделенные знаком «минус». Примеры использования команды `\cline` будут даны ниже (см. с. 219 и 222).

Абзацы в графах таблицы. Иногда требуется, чтобы в графе таблицы стояла не строка, а абзац текста, переносы и разрывы строк в котором находятся автоматически. Чтобы этого добиться, надо в преамбуле вместо буквы `l`, `c` или `r`, описывающей структуру колонки, написать `p{...}`, где вместо многоточия должна быть указана ширина колонки (в TeX'овских единицах — см. с. 26). Вот как можно представить в виде таблицы известную шутку М. М. Жванецкого:

Я видел раков	
Вчера:	Сегодня:
Маленькие, но по три рубля, но очень маленькие, но по три, но очень маленькие.	Большие, но по пять рублей, но большие, но по пять рублей, но очень большие, но по пять.

Исходный текст был таков:

```
\begin{tabular}{|p{5cm}|p{5cm}|}
\hline
\multicolumn{2}{|c|}{\large\textbf{Я видел раков}}\\
\hline
Вчера: & Сегодня: \\
Маленькие, но по три рубля, но очень
маленькие, но по три, но очень маленькие.
&
Большие, но по пять рублей, но большие,
по пять рублей, но очень большие,
но по пять.\\
\hline
\end{tabular}
```

По умолчанию абзацы в графах таблицы печатаются выровненными, но без абзацного отступа; если абзацный отступ нужен, начните абзац с установки необходимого значения параметра `\parindent`; если выравнивание не нужно, дайте команду `\raggedright`; одним словом, вы можете проделывать с этими абзацами все манипуляции, описанные в разд. III.6.

Из этого правила есть одно важное исключение: для принудительного разрыва строки в абзацах, являющихся графами таблицы, вместо команды `\\` надо использовать команду `\tabularnewline` (команда `\\` в окружениях `tabular` и `array` означает «перейти к новой строке таблицы»).

Что такое колонка? При работе с линованными таблицами возникает вопрос, как LaTeX понимает слова «одна колонка»? Пусть, например, преамбула таблицы имеет вид `||c|ll||r|l|l`, и мы в одной из строк написали, скажем,

```
Что-то&\multicolumn{1}{r}{Что-то еще}&&&И еще&Еще\\
```

Напечатается ли в этой графе вертикальный отрезок между первой и второй колонками? Другой пример: пусть в таблице с той же преамбулой какая-то из строк имеет вид

```
Слово & Еще слово & Еще одно\\
```

(стало быть, заканчивается эта строка преждевременно); сколько вертикальных отрезков будет напечатано в конце этой строки? Ответ таков. Преамбула делится на части, соответствующие колонкам. Если в преамбуле присутствуют только буквы `l`, `c`, `r` или `p`, то каждая такая часть — это просто соответствующая буква (`p` — вместе с выражением после нее: `p{...}`). Если же, кроме этого, в преамбуле присутствуют вертикальные черточки между буквами или так называемые at-выражения (о них речь пойдет ниже), разделение преамбулы на колонки происходит по таким правилам:

- в каждой из колонок присутствует одна и только одна из букв `l`, `c`, `r` или `p` (последняя — вместе с выражением `{...}`);
- каждая колонка, кроме, возможно, первой, начинается с буквы.

В нашем примере, в частности, колонки устроены так:

Преамбула	`||c|ll||r|l|l`
Первая колонка	`||c|`
Вторая колонка	`l`
Третья колонка	`l||`
Четвертая колонка	`r|`
Пятая колонка	`l|`
Шестая колонка	`l`

Поэтому в конце графы таблицы с такой преамбулой, оборванной после третьей колонки, будут напечатаны два вертикальных отрезка, поскольку они принадлежат третьей колонке. А если на месте второй графы такой таблицы написано `Что-то&\multicolumn{1}{r}{Что-то еще}`, то вертикальный отрезок между первой и второй колонками также будет напечатан: этот отрезок является принадлежностью первой колонки, и команда `\multicolumn`, меняющая оформление второй колонки, отменить его не может.

3. Примеры

В этом разделе мы приведем различные примеры верстки сложных таблиц с помощью LaTeX'а. По ходу дела будет рассказано и о некоторых изысканных возможностях окружений `tabular` и `array`, о которых до сих пор речи не было. Кое-где в этом разделе мы будем предполагать, что читатель знаком со средствами математического набора, описанными в гл. II.

Таблица VI.2.

существительные формы		прилагательные формы
мой	le mien, la mienne les miens, les miennes	mon, ma, mes
твой	le tien, la tienne les tiens, les tiennes	ton, ta, tes
его, ее, свой	le sien, la sienne les siens, les siennes	son, sa, ses
наш	le nôtre, la nôtre, les nôtres	notre, nos
ваш	le vôtre, la vôtre, les vôtres	votre, vos
их, свой[1]	le leur, la leur, les leurs	leur, leurs

[1] Лишь в значении принадлежности 3-му лицу.

Наш первый пример — таблица французских притяжательных местоимений, взятая из русско-французского словаря акад. Л. В. Щербы (табл. VI.2), которую мы задали в LaTeX'е так:

```
{\small
\begin{tabular}{cll}
\multicolumn{2}{c}{существительные формы}
    &  прилагательные формы\\[\smallskipamount]
мой & $\left\{ \begin{tabular}{l}
         le mien, la mienne\\les miens, les miennes\\
       \end{tabular} \right. $
    & mon, ma, mes\\[\bigskipamount]
твой & $\left\{ \begin{tabular}{l}
         le tien, la tienne\\les tiens, les tiennes\\
       \end{tabular}\right. $
    & ton, ta, tes\\[\bigskipamount]
его, ее, свой &
      $\left\{ \begin{tabular}{l}
         le sien, la sienne\\les siens, les siennes\\
       \end{tabular}\right. $
    & son, sa, ses\\[\bigskipamount]
наш & le n\^otre, la n\^otre, les n\^otres & notre, nos\\
ваш & le v\^otre, la v\^otre, les v\^otres & votre, vos\\
их, свой$^1$ &  le leur, la leur, les leurs
    &  leur, leurs\\ \cline{1-1}
\multicolumn{3}{l}{$^1$\rule{0pt}{11pt}\footnotesize
    Лишь в значении принадлежности 3-му лицу.}\\
\end{tabular}
}
```

Разберем, как устроена эта таблица. Как явствует из ее преамбулы `cll`, она состоит из трех колонок, из которых левая центрирована, а две другие прижаты влево. Соответственно, три последние графы набраны совершенно бесхитростно. Заголовок таблицы сделан с помощью команды `\multicolumn`; команда `\\`, завершающая первую строку таблицы, имеет необязательный аргумент; это сделано, чтобы отодвинуть заголовок по вертикали от остальной части таблицы.

Рассмотрим теперь, как устроена вторая графа (начинающаяся с местоимения «мой»). Текст

$$\left\{\begin{array}{l}\text{le mien, la mienne}\\ \text{les miens, les miennes}\end{array}\right.$$

образует в нашей таблице одну «запись» (часть таблицы, расположенную на пересечении графы и колонки). Чтобы получить фигурную скобку требуемого (и неизвестного нам заранее) размера, мы воспользовались командами `\left` и `\right`, применяемыми при наборе формул (см. разд. II.2.5). Так как эти команды вне формул использовать нельзя, нам пришлось оформить этот фрагмент текста как формулу. Между `\left\{` и `\right.` стоит, как водится, та формула, по размеру которой получается фигурная скобка, заданная командой `\left\{` — в нашем случае эта «формула» является фрагментом текста, задаваемым с помощью еще одного окружения `tabular` (с преамбулой `l`). Команды `\\`, завершающие первые три графы основной части таблицы, имеют необязательные аргументы, задающие дополнительные вертикальные пробелы после этих граф (иначе фигурные скобки будут упираться друг в друга и портить вид таблицы).

К местоимению «свой» в последней строке таблицы дана сноска. Знак сноски реализован нами опять же как математическая формула — верхний индекс 1 к «пустой формуле»; текст сноски реализован как графа таблицы, охватывающая все три колонки (с помощью команды `\multicolumn`). Команда `\footnotesize` задает размер шрифта, используемый в обычных сносках (см. разд. III.5). Линия, отделяющая сноску от остальной части таблицы, реализована с помощью команды `\cline`. Наконец, посмотрим, как задана цифра 1 в самом тексте сноски. Вместо ожидаемого `$^1$` написано вот что:

```
$^1$\rule{0pt}{11pt}
```

Как объясняется в разд. III.10, команда `\rule` задает в данном случае невидимый символ, занимающий по вертикали 11 пунктов и не занимающий места по горизонтали. Мы поставили этот невидимый символ в качестве подпорки: без нее горизонтальная черта соприкасалась бы с цифрой 1.

Вся таблица в целом набрана мелким шрифтом (иначе она не помещалась на страницу).

Таблица VI.3.

Понедельник	8^{30}–15	Обед	11–12
Вторник	12–19	Обед	15–16
Среда	10–17	Обед	12^{30}–13^{15}
Четверг	9–17	Обед	12–13
Пятница	11–16	Обед	–
Суббота	8–14	Обед	11–12

Следующий пример (табл. VI.3) — расписание работы одной химчистки. Для него исходный текст выглядит так:

```
\begin{tabular}{lr@{--}l@{\qquad Обед\quad}r@{--}l}
Понедельник & $8^{30}$ & 15 & 11 & 12 \\
Вторник     & 12 & 19 & 15 & 16 \\
Среда       & 10 & 17 & $12^{30}$ & $13^{15}$ \\
Четверг     & 9 & 17 & 12 & 13 \\
Пятница     & 11 & 16 & &\\
Суббота     & 8 & 14 & 11 & 12\\
\end{tabular}
```

В преамбуле тут используется конструкция, с которой мы пока не встречались. Объясним, что́ она делает.

До сих пор мы говорили, что в преамбуле каждая колонка таблицы может обозначаться символом `l`, `c`, `r` или `p{...}` , а по краям или между колонками могут еще стоять вертикальные черточки `|`, обозначающие разделительные вертикальные линейки. Это, однако, не вся правда. В качестве разделителя колонок (а также с краев) в преамбуле может быть использовано еще и так называемое «at-выражение»[1]: символ `@`, непосредственно после которого в фигурных скобках записан какой-то текст, возможно, с TEX'овскими командами. В таблице этот текст будет вставлен между соответствующими колонками во всех строках (если, разумеется, формат какой-то графы таблицы не был изменен командой `\multicolumn`). Мы использовали at-выражение трижды: два раза для вставки тире и один раз — для слова «Обед». А зачем же нам понадобились команды `\qquad` и `\quad` вокруг этого слова? Дело в том, что между колонками, разделенными at-выражением, *не* вставляется дополнительный интервал, которым LATEX разделяет колонки в

[1]Мы выбрали для него такое название, поскольку официально символ `@` называется «коммерческое at»; неофициально этот символ называют самыми разными именами, от «собаки» до «блямбы».

таблицах, созданных с помощью окружений `tabular` или `array`: именно поэтому тире между часом открытия химчистки и часом ее закрытия плотно прилегает к обоим числам. Слово «Обед», однако же, совсем не должно вплотную прилегать к началу обеденного перерыва, поэтому промежуток нужно создать самому, и проще это сделать один раз внутри все того же at-выражения, чем писать `\quad` для каждого рабочего дня.

Иногда at-выражение имеет смысл применять даже в виде `@{}`: между колонками при этом ничего не вставится, но зато дополнительный интервал между колонками, разделенными этим выражением, будет подавлен. Если написать `@{}` в преамбуле перед символом, обозначающим первую колонку, или после символа, обозначающего последнюю колонку, то будет подавлен дополнительный интервал, вставляемый перед первой или после последней колонки (это может помочь, если таблица немного не помещается на страницу по ширине).

Иногда интервал между колонками, автоматически устанавливаемый окружением `tabular` или `array`, является неудачным (ниже мы разберем соответствующий пример). В этом случае можно самостоятельно установить для него подходящее значение. Для этого надо присвоить новое значение параметру `\tabcolsep` для окружения `tabular` или `\arraycolsep` для окружения `array` (см. разд. I.2.9 по поводу параметров). По обе стороны от каждой колонки таблицы добавляется пробел размером `\tabcolsep` (соответственно `\arraycolsep`). Стало быть, значение этих параметров — *половина* расстояния между соседними колонками.

Наряду с расстоянием между колонками можно менять толщину линеек в линованных таблицах (обозначается `\arrayrulewidth`; относится этот параметр как к `array`, так и к `tabular`), а также расстояние между соседними линейками — это расстояние обозначается `\doublerulesep`, и оно также относится в равной мере к `array` и к `tabular`.

Теперь разберем обещанный пример, в котором приходится менять заданное по умолчанию расстояние между колонками. Посмотрите на такую формулу:

$$\begin{array}{r@{}r@{}r|l}
x^2 & {}+2x & {}-12 & x+5 \\ \cline{4-4}
\underline{x^2} & \underline{{}+5x} & & x-3 \\
 & -3x & {}-12 & \\
 & \underline{-3x} & \underline{{}-15} & \\
 & & 3 &
\end{array}$$

Она была создана с помощью следующих LaTeX'овских команд:

```
$$
\arraycolsep=0.05em
\begin{array}{rrr@{\,}r|r}
x^2&{}+2x&{}-12&&\,x+5\\
\cline{5-5}
x^2&{}+5x&&&\,x-3\\
\cline{1-2}
&{}-3x&{}-12\\
&{}-3x&{}-15\\
\cline{2-3}
&&3
\end{array}
$$
```

Сразу же скажем, зачем нам понадобилось менять `\arraycolsep`: без этого интервалы между слагаемыми в каждой строке выходили непомерно большими. А теперь разберем исходный текст подробнее. Начнем с преамбулы `rrr@{\,}r|r`. В ней первые три колонки отведены под слагаемые, наподобие x^2, $+2x$ или -12; пятая колонка предназначена для делителя и частного ($x+5$ и $x-3$), а вертикальная черточка в преамбуле перед буквой `r`, задающей пятую колонку — для вертикального отрезка, входящего в состав «уголка». С другой стороны, в четвертой колонке нет вообще никакого текста: между третьим и четвертым знаками `&` ни в одной строке ничего не написано. Эту пустую колонку мы создали для того, чтобы вертикальный отрезок не пошел ниже, чем нужно: без нее с преамбулой `rrr|r` вертикальный отрезок относился бы к четвертой колонке (в соответствии с правилами на с. 218), и в результате третья строка закончилась бы вертикальным отрезком, что нам совсем ни к чему.

Осталось заметить, что пары долларов, ограничивающие выключную формулу, заодно ограничивают и группу, так что по окончании формулы закончится и группа, и старое значение `\arraycolsep` восстановится автоматически.

Наш последний пример использования окружения `tabular` связан с проблемой, с которой мы столкнулись на с. 215: как ликвидировать разрыв в вертикальных линейках, получающийся, если в линованной таблице написать две команды `\hline` подряд? Первое, что приходит в голову, — создать еще одну графу в таблице, в которой поместить только невидимую линейку высотой, скажем, 2 пункта; казалось бы, тогда горизонтальные линейки будут на расстоянии 2 пункта друг от дружки, а вертикальные линейки не будут прерываться. Результат, однако, получается совершенно неудовлетворительный:

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

```
\begin{tabular}{|c|c|}
\hline
Северо-Запад & Северо-Восток\\
\hline \rule{0pt}{2pt}&\\
\hline
Юго-Запад & Юго-Восток\\
\hline
\end{tabular}
```

Чтобы понять, в чем тут дело, нам придется обсудить, каким образом LaTeX собирает таблицу из отдельных строк.

Таблицы, созданные с помощью окружения `tabular` или `array`, собираются из строк, которые вплотную приставляются друг к другу. При этом, чтобы расстояния между строками были одинаковыми, в каждую строку предварительно вставляется невидимая линейка (именно, линейка, создаваемая командой `\strut`). Из-за этой линейки расстояние между горизонтальными отрезками оказалось слишком большим, а наша линейка высотой в 2 пункта LaTeX'у не помогла: ведь `\strut` все равно выше! Чтобы обойти эту трудность, в LaTeX'е предусмотрен способ отменить автоматическую постановку `\strut`'ов во всех строках таблицы. Именно, для этого надо написать (*не* внутри окружения `tabular` или `array`!) так:

```
\renewcommand{\arraystretch}{0}
```

Что такое `\renewcommand`, мы будем обсуждать в гл. VII, а пока давайте воспринимать этот рецепт догматически. Скажем только, что, во-первых, если эта команда была дана внутри группы, то по выходе из группы ее действие отменяется, и, во-вторых, в явном виде восстановление режима, когда в каждую строку таблицы вставляется `\strut`, достигается с помощью команды

```
\renewcommand{\arraystretch}{1}
```

Теперь уже легко добиться желаемого эффекта; надо только не забыть поставить в нужные строки команду `\strut` в явном виде, коль скоро автоматически это теперь не делается. Итак, таблица

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

набирается следующим образом:

```
{\renewcommand{\arraystretch}{0}%
\begin{tabular}{|c|c|}
```

```
\hline \strut Северо-Запад & Северо-Восток\\
\hline \rule{0pt}{2pt}&\\
\hline \strut Юго-Запад & Юго-Восток\\
\hline
\end{tabular}%
}
```

Знаки процента в конце некоторых строк мы поставили, чтобы концы этих строк не воспринимались как пробелы (на самом деле в данной ситуации вреда от пробелов не было бы). Закрывающая фигурная скобка в последней строке закрывает группу, из которой была дана команда `\renewcommand`.

Если граф в таблице много, то, возможно, вам не захочется много раз писать `\strut`. В этом случае можно включить эту команду в преамбулу с помощью at-выражения. Возможный вариант такой:

```
{\renewcommand{\arraystretch}{0}%
\begin{tabular}{|@{\strut\hspace{\tabcolsep}}c|c|}
\hline
Северо-Запад & Северо-Восток\\
\hline
\multicolumn{1}{|c|}{\rule{0pt}{2pt}}&\\
\hline
Юго-Запад & Юго-Восток\\
\hline
\end{tabular}
}
```

Если бы в аргументе at-выражения не был указан горизонтальный пробел размером `\tabcolsep`, то левая вертикальная линейка была бы напечатана вплотную к тексту (потому что at-выражение подавляет автоматически вставляемый горизонтальный пробел); заметим также, что теперь, когда `\strut` включен в at-выражение, нам пришлось воспользоваться командой `\multicolumn`, чтобы этот `\strut` не попал и в ту строку, где мы так старались от него избавиться.

Описанный способ набрать таблицу с удвоенной горизонтальной линейкой — не единственный. Если подключить описываемый в следующем разделе стилевой пакет `hhline`, то можно это сделать, и не играя с командой `\arraystretch`.

Можно не только отменять автоматическое добавление `\strut`'а в строки таблицы, но и изменять его высоту. Например, если мы хотим, чтобы размер этой линейки увеличился (во всех строках) в 3.7 раза, можно написать:

```
\renewcommand{\arraystretch}{3.7}
```

(вместо десятичной точки можно поставить и десятичную запятую).

4. Дополнительные возможности

Выше мы описывали возможности печати таблиц, доступные «чистому» LaTeX'у (без подключения дополнительных стилевых пакетов). Ряд стилевых пакетов, входящих в комплект поставки LaTeX'а, позволяет добиться дополнительных интересных эффектов.

4.1. Пакет array

В этом разделе мы рассказываем о различных мелких (но нередко полезных) дополнительных возможностях, открывающихся при подключении стилевого пакета `array`.

Итак, предположим, что этот пакет подключен. Что нового вы сможете сделать?

При пользовании командой `\hline` горизонтальные линейки иногда слишком плотно примыкают к тексту (особенно если текст содержит прописные буквы). В «чистом» LaTeX'е для борьбы с этим надо либо писать

```
\renewcommand{\arraystretch}...
```

либо вставлять в каждую строку по дополнительной распорке. При подключении пакета `array` появляется и более простой способ: надо присвоить ненулевое значение параметру `\extrarowheight`. Это — величина, которая добавляется к высоте каждой строки таблицы. Этому параметру можно присваивать значения так же, как и любому другому параметру со значением длины (см. с. 26); по умолчанию его величина равна нулю, для лучшего отделения линеек от текста хорошо присвоить ему значение 2–3 пункта.

Если вы пользовались окружением `tabular` с необязательным аргументом `t`, задающим выравнивание таблицы как «буквы» по верхней строке, то могли обратить внимание, что это выравнивание нарушается, если таблица начинается с горизонтальной линейки:

а	б
в	г

а	б
в	г

```
\begin{tabular}[t]{rr}
а & б \\в & г
\end{tabular}
\begin{tabular}[t]{|rr|}
\hline а & б \\в & г\\
\hline
\end{tabular}
```

Чтобы выравнивание происходило не по линейке, а по первой строке текста, надо задать верхнюю линейку командой `\firsthline`, а

а	б	в	г				
д	е	ж	з	а	б	в	г
				д	е	ж	з

а	б	в	г	а	б	в	г
д	е	ж	з	д	е	ж	з

```
\begin{tabular}[t]{rrrr}
а & б & в & г \\
д & е & ж & з
\end{tabular}
\begin{tabular}[t]{|rrrr|}
\hline а & б & в & г \\
д & е & ж & з\\ \hline
\end{tabular}\\[1in]
\begin{tabular}[t]{rrrr}
а & б & в & г \\
д & е & ж & з
\end{tabular}
\begin{tabular}[t]{|rrrr|}
\firsthline а & б & в & г \\
д & е & ж & з\\ \hline
\end{tabular}
```

Рис. VI.1.

не `\hline`, как видно из рис. VI.1. Аналогично, чтобы при пользовании `tabular` с необязательным аргументом `b` выравнивание таблицы как целого шло по нижней строке текста, а не по нижней линейке, надо нижнюю горизонтальную линейку задать командой `\lasthline`, а не `\hline`.

Как мы знаем, в преамбуле окружения `tabular` (а также `array`) могли стоять буквы `l`, `r`, `c` или выражение `p{...}`, обозначающие тип колонки, а между ними — вертикальные черточки или at-выражения. Пакет `array` добавляет кое-что к этому списку.

Во-первых, при подключении этого пакета в преамбуле, наряду с выражением `p{...}`, можно пользоваться выражениями `m{...}` и `b{...}`. Как и `p{...}`, они указывают, что в колонке стоит абзац текста ширины, заданной в фигурных скобках. Однако в графах абзац, заданный с помощью `b{...}`, выравнивается по своей нижней строке, абзац, заданный с помощью `m{...}` — по середине своей высоты, а абзац, заданный с помощью `p{...}`, всегда выравнивался по своей верхней строке. См. рис. VI.2.

Наряду с at-выражениями, пакет `array` позволяет использовать в преамбуле еще и !-выражения. Именно, между буквами, обозначающими колонки, можно, наряду с вертикальными черточками и at-выражениями, написать `!{...}`, где на месте точек стоят какие-то TeX'овские команды и/или текст. Эта конструкция оказывает то же

		Все выше, и
	Все выше, и	выше, и вы-
Все выше, и	выше, и вы-	ше!
выше, и вы-	ше!	
ше!		

```
\begin{tabular}{p{.9in}m{.9in}b{.9in}}
Все выше, и выше, и выше! &
Все выше, и выше, и выше! &
Все выше, и выше, и выше!\\
\end{tabular}
```

Рис. VI.2.

действие, что и at-выражение, но при этом, в отличие от at-выражения, не подавляет интервал между колонками. Поэтому !-выражение удобно использовать для увеличения интервала между колонками: в таблице с преамбулой

```
{rc!{\hspace{2pt}}cl}
```

интервал между двумя центрированными колонками будет увеличен на два пункта.

Другое возможное применение !-выражений — печать линованных таблиц, в которых вертикальные линейки, разделяющие колонки, имеют разную ширину. Если, например, мы хотим, чтобы какая-то из вертикальных линеек имела ширину `1pt`, а не `\arrayrulewidth`, надо в преамбуле вместо вертикальной черточки `|`, обозначающей эту линейку, написать `!{\vrule width 1pt\relax}` (см. с. 153 по поводу команды `\vrule`).

Наконец, еще одна интересная возможность, предоставляемая пакетом `array`, — это автоматическая вставка TEX'овских команд в начале и/или конце колонки. Именно, если в преамбуле непосредственно перед любой из букв `l`, `c`, `r`, `t`, `m` или `b`, обозначающих тип колонки, вставить выражение

`>{`*команды*`}`

то *команды* будут автоматически добавляться в начало соответствующей колонки. Это может пригодиться, если вам нужно сменить шрифт в одной из колонок:

Генрих II	*1519–1559*
Генрих III	*1551–1589*
Генрих IV	*1553–1610*

```
\begin{tabular}{|>{Генрих }l|>{\slshape}l|}
\hline
II & 1519--1559\\III & 1551--1589\\
IV & 1553--1610\\ \hline
\end{tabular}
```

Можно также *после* буквы, обозначающей тип колонки, или после конца `p`-, `m`-, или `b`-выражения, написать

`<{`*команды*`}`

чтобы *команды* были добавлены в конец колонки. Эта конструкция полезна, если какие-то из колонок в таблице, оформленной как `tabular`, должны набираться в математическом режиме — достаточно поставить в начале и конце этой колонки по знаку доллара. Пример:

квадрат суммы	$(x+y)^2$
квадрат разности	$(x-y)^2$
сумма кубов	x^3+y^3

```
\begin{tabular}{l>{$}l<{$}}
квадрат суммы & (x+y)^2\\
квадрат разности & (x-y)^2\\
сумма кубов & x^3+y^3
\end{tabular}
```

4.2. Пересечения линеек

Возможностей окружения `array` вполне хватает для печати простейших линованных таблиц, но в более сложных случаях возникают проблемы (см. пример на с. 215). Если подключить стилевой пакет `hhline`, работа с линованными таблицами облегчается.

Итак, предположим, что этот пакет подключен. Тогда для задания горизонтальных линеек становится доступной, наряду с уже известными `\hline` и `\cline`, новая команда `\hhline`, в аргументе которой описывается как сама линейка, так и ее пересечения с вертикальными линейками. Вот первый пример ее использования:

А	Б	В	Г
Д	Е	Ж	З

```
\begin{tabular}{|c|cc|c|}
\hline А & Б & В & Г\\
\hhline{|=|~~|-|}
Д & Е & Ж & З\\ \hline
\end{tabular}
```

Аргумент команды `\hhline` устроен следующим образом. Во-первых, в нем сказано, что на территории первой колонки линейка должна быть двойной (символ `=`), на территории второй и третьей колонок линейки не должно быть вовсе (символ `~` — «тильда»), а на территории четвертой колонки линейка должна быть одинарной (символ `-`). Если в таблице n колонок, то в аргументе `\hhline` должны присутствовать n символов `-`, `=` или `~`, имеющих тот же смысл, что и выше.

Между этими символами, описывающими поведение линейки внутри колонок, расположены символы, описывающие пересечения горизонтальной линейки с вертикальными. В нашем примере это были вертикальные черточки `|`; кроме них, для задания информации о пересечениях линеек можно использовать символы `:`, `#`, а также буквы `t` и `b`.

Какие именно пересечения линеек можно получить с их помощью, видно из следующей таблицы:

На печати	├	┤	┼, ┬, ┴	╞	╡	╪, ╤, ╧
В аргументе `\hhline`	`|-`	`-|`	`-|-`	`|=`	`=|`	`=|=`
На печати	╞	╡	╪, ╤, ╧	╟	╢	╫
В аргументе `\hhline`	`:=`	`=:`	`=:=`	`||-`	`-||`	`-||-`
На печати	╠	╣	╬	╠	╣	╬, ╦, ╩
В аргументе `\hhline`	`|:=`	`=:|`	`=::=`	`#=`	`=#`	`=#=`
На печати	╔	╚	╗	╝	╦	╩
В аргументе `\hhline`	`|t:=`	`|b:=`	`=:t|`	`=:b|`	`=:t:=`	`=:b:=`

Вот пример таблицы, в которой используются эти возможности команды `\hhline`:

1	2	3	4
5	6	7	8
А	Б	В	Г
Д	Е	Ж	З

```
\begin{tabular}{||cc||cc||}
\hhline{|t:==:t:==:t|}
1 &2 &3 &4\\5 &6 &7 &8\\
\hhline{#==::==||}
А & Б & В & Г\\
\hhline{||--||~~}
Д & Е & Ж & З\\
\hhline{|b:==:b:==:b|}
\end{tabular}
```

Подчеркнем, что команда `\hhline` обрабатывает пересечения линеек независимо от того, какие вертикальные линейки заданы в преамбуле. Забота о том, чтобы аргумент `\hhline` был согласован с преамбулой, лежит на вас.

4.3. Таблицы, простирающиеся на несколько страниц

Как уже отмечалось, окружения `array` и `tabular` рассматриваются LaTeX'ом как одна большая буква, и потому не разбивается по страницам. Можно, однако, создавать таблицы, в которых и разбиение на страницы, и определение ширины колонок происходит автоматически. Для этого надо подключить стилевой пакет `longtable` и использовать окружение `longtable`. Как и окружение `tabular`, оно принимает один обязательный параметр — преамбулу (устроенную точно так же, как у `tabular`); внутри окружения действуют в точности те же правила записи текста, что и в окружении `tabular` (в частности, допустимы команды `\hline`, `\cline` и `\multicolumn`). Разница с окружением `array` в том, что получаемая при этом таблица может занимать несколько страниц и иметь произвольную длину.

Опишем теперь некоторые особенности окружения `longtable`, связанные с разбиением на страницы.

Во-первых, после первого запуска LaTeX'а колонки таблицы, определенной как `longtable`, могут оказаться невыровненными (это связано с тем, что при первом проходе LaTeX читает такую таблицу не целиком, а по кускам, и выравнивает эти куски независимо друг от друга). Чтобы добиться выравнивания, надо запустить LaTeX еще раз-другой (если есть необходимость в повторном запуске, об этом будет выдано предупреждение).

Во-вторых, вы имеете возможность сделать так, чтобы заголовок таблицы повторялся на каждой новой странице, на которой таблица продолжается. Для этого надо оформить заголовок в виде строки или группы строк (и/или команд `\hline`), и при этом последнюю из этих строк надо завершить не командой `\\`, а командой `\endhead`. Обычно, впрочем, повторяющийся заголовок не идентичен тому, который ставят в самом начале таблицы. Отдельный заголовок для начала таблицы также оформляют в виде одной или нескольких строк, последняя из которых завершается командой `\endfirsthead`. Кроме того, можно предусмотреть специальную группу строк, которая будет ставиться на каждой странице *внизу* таблицы — для этого надо записать строку или группу строк, завершив последнюю из них не командой `\\`, как обычно, а командой `\endfoot`. Можно также предусмотреть отдельную группу строк, которая ставится внизу таблицы только на последней из занимаемых таблицей страниц. Для этого надо вместо `\endfoot` написать `\endlastfoot`. Группы строк, завершающиеся командами `\endhead`, `\endfirsthead`, `\endfoot` или `\endlastfoot`, должны стоять *в начале* окружения `longtable`. Вот пример использования этих средств:

Итоги собачьей выставки

Кличка	Пол	Порода	Оценка
1	2	3	4
Алекс	кобель	миттельшнауцер	отл.
Ассоль	сука	ирландский терьер	отл.
Бима	сука	кавказская овчарка	оч. хор.
Велли	сука	ризеншнауцер	отл.
Грант	кобель	ризеншнауцер	отл.
Джек	кобель	дог	отл.
Джерри	кобель	ирландский терьер	отл.
Жорж	кобель	бриар	оч. хор.
Зента	сука	ирландский терьер	оч. хор.
Клия	сука	бладхаунд	отл.

Продолжение на следующей странице

1	2	3	4
Малыш	кобель	метис	отл.
Найт	кобель	ризеншнауцер	отл.
Одри	сука	ризеншнауцер	отл.
Ричард	кобель	ирландский сеттер	отл.
Сэнди	кобель	немецкая овчарка	отл.
Тима	кобель	миттельшнауцер	отл.
Чапик	кобель	метис	отл.
Чара	сука	ротвейлер	оч. хор.
Эмир	кобель	эрдельтерьер	оч. хор.

Этой таблице соответствовал такой исходный текст:

```
\begin{longtable}{|l|l|l|l|}
\multicolumn{4}{c}{Итоги собачьей выставки}\\
\hline
Кличка & Пол & Порода & Оценка\\
\hline
1&2&3&4\\
\hline\endfirsthead
\hline
1 & 2 & 3 & 4\\
\hline\endhead
\hline
\multicolumn{4}{c}{\textit{Продолжение на следующей странице}}
\endfoot
\hline\endlastfoot
Алекс & кобель & миттельшнауцер & отл.\\
...
Эмир & кобель & эрдельтерьер & оч. хор.\\
\end{longtable}
```

В таблице, оформленной с помощью `longtable`, можно явно указать место разрыва страницы с помощью `\newpage` (а также `\pagebreak` или `\nopagebreak`). Эти команды должны следовать непосредственно после `\\` (можно с новой строки).

При пользовании пакетом `longtable` предусмотрена возможность автоматической нумерации таблиц, созданных окружением `longtable`. Для этого используется та же команда `\caption`, что и в окружениях `table` или `picture`, но пользоваться ей надо чуть по-другому: после этой команды необходимо поставить `\\`, `\endhead` или `\endfirsthead` (или `\endfoot`,...).

При этом автоматически нумеруемые таблицы можно обычным образом метить с помощью `\label` (и потом ссылаться на эти места с помощью `\ref`), но необходимо соблюдать два ограничения:

- метку нельзя ставить в строки, которые появятся на печати более одного раза (благодаря повторению заголовков на других страницах);
- с команды `\label` не должна начинаться ни одна графа таблицы.

На автоматически нумерующийся заголовок выделяется по умолчанию 4 дюйма. Чтобы изменить этот размер, надо присвоить соответствующее значение параметру `\LTcapwidth`.

В отличие от окружения `tabular`, таблица в окружении `longtable` не рассматривается как одна большая буква, а сразу располагается между абзацами, по умолчанию — по центру. Чтобы таблица была прижата к правому краю, надо указать у окружения `longtable` необязательный аргумент `r` (в квадратных скобках, между `\begin{longtable}` и преамбулой); необязательный аргумент `l` даст таблицу, прижатую влево.

Глава VII

Создание новых команд

Средства LaTeX'а, описываемые в этой главе, позволяют сократить число нажатий на клавиши при наборе сложных текстов. Именно, мы расскажем, как создавать новые команды (или, если угодно, сокращенные обозначения), заменяющие собой длинные фрагменты из текста и TeX'овских команд. Официально такие новые команды называются макроопределениями, а в разговорной речи — макросами.

1. Макроопределения

1.1. Команды без аргументов

Начнем с примера. Пусть вы пишете текст, в котором регулярно встречается математический значок $\stackrel{\mathrm{def}}{=}$ (он означает «равно по определению»). Пользуясь тем, что вы узнали из гл. II, нетрудно понять, что генерируется этот значок внутри математической формулы такой последовательностью команд:

```
\stackrel{\mathrm{def}}{=}
```

Часто писать такой длинный набор команд утомительно. Вот бы в LaTeX'е была предусмотрена команда, скажем, `\eqdef`, генерирующая символ бинарного отношения $\stackrel{\mathrm{def}}{=}$! Правда, такой команды нет, но мы ее можем создать. Для этого следует написать так:

```
\newcommand{\eqdef}{\stackrel{\mathrm{def}}{=}}
```

После того как TeX прочтет эту строку, он всюду, встречая команду `\eqdef`, будет реагировать точно так же, как если бы он видел текст `\stackrel{\mathrm{def}}{=}`. Например, формула $x^2 \stackrel{\mathrm{def}}{=} x \cdot x$ теперь получается так:

```
x^2\eqdef x\cdot x
```

Новая команда TEX'а, которую мы определили, называется макросом (еще говорят: макроопределение, макрокоманда, макро). Рассмотрим точные правила для создания макросов средствами LaTeX'а.

Для создания макросов используется команда `\newcommand`. Эта команда имеет два обязательных аргумента. Первый из них — имя, которое вы придумали для вашего макроса. Имена макросов должны подчиняться тем же правилам, что имена TEX'овских команд (см. разд. I.2.3): либо backslash и после него одна не-буква, либо backslash и после него — последовательность букв. Второй обязательный аргумент команды `\newcommand`, называемый «замещающим текстом», сообщает TEX'у смысл макроса: на этот текст ваш макрос будет замещаться в процессе трансляции (как говорят, макрос будет «разворачиваться»).

При пользовании командой `\newcommand` нельзя в качестве имени макроса выбирать имя уже существующей команды или окружения (если вы попробуете так сделать, LaTeX выдаст сообщение об ошибке).

Во втором аргументе команды `\newcommand` (иными словами, в «замещающем тексте») вместе с каждой открывающей фигурной скобкой должна присутствовать соответствующая ей закрывающая[1], так что определения наподобие

```
\newcommand{\nachatkursiv}{{\itshape}
\newcommand{\konchitkursiv}{}}
```

приведут в лучшем случае к сообщению об ошибке (и желаемого эффекта не дадут). Если вам кажется, что такие ограничения стеснительны, можете изучить по книге [2], как их обходить; для большинства практических целей возможности создания макроопределений, предоставляемые LaTeX'ом, вполне достаточны.

К сожалению, некоторые русификации TEX'а не позволяют использовать в именах макросов русские буквы.

Еще одно ограничение: имя новой команды не должно начинаться на `end`.

Наконец, в замещающем тексте макроопределения нельзя пользоваться командой `\verb` или окружением `verbatim`.

Если команда `\newcommand` дана внутри группы, то смысл определяемой ею новой команды будет забыт TEX'ом по выходе из группы. Если новая команда определяется в преамбуле, то, естественно, она будет понятна TEX'у на протяжении всего документа.

Давайте теперь разберем несколько примеров, обращая внимание на типичные ошибки.

[1]Фигурные скобки, входящие в состав команд `\{` и `\}`, в счет при этом не идут.

Макросы хороши как средство скорописи. Например, если в вашем тексте часто встречается знак $\bigtriangleup$, то вам может надоесть все время писать длинную команду `\bigtriangleup`. Коли так, придумайте сокращенное обозначение (скажем, `\btu`), напишите в преамбуле

```
\newcommand{\btu}{\bigtriangleup}
```

и вы сможете писать формулы наподобие

$(A \bigtriangleup B) \cap C = (A \cap C) \bigtriangleup (B \cap C)$

```
$(A\btu B)\cap C=
(A\cap C)\btu (B\cap C)$
```

На с. 50 было рассказано, что делать, чтобы создать согласующееся с нашими традициями обозначение для тангенса. Теперь мы понимаем, что это был пример макроопределения (если вам интересно знать, что значит команда `\mathop` в замещающем тексте, загляните в разд. II.5.4).

В гл. II вы найдете массу других примеров громоздких конструкций, для которых имеет смысл создать макросы.

А теперь — пример типичной ошибки. Пусть в тексте, который вы набираете, регулярно встречаются фразы наподобие

> Подмногообразия проективного пространства $\mathbf{P}^n$ — основной объект изучения алгебраической геометрии,

и пусть для сокращения письма вы написали в преамбуле

```
\newcommand{\Pn}{$\mathbf P^n$}
```

Теперь можно писать, например, так:

```
... пространства~\Pn{} --- основной объект...
```

Однако для набора формулы $x \in \mathbf{P}^n$ написать `$x\in\Pn$` не удастся: появится сообщение о том, что символ `^` и команда `\mathbf` преступно употреблены вне математической формулы. Причина проста: TEX исправно подставляет вместо `\Pn` тот «замещающий текст», который вы ему сообщили во втором аргументе команды `\newcommand`. В результате этого при развертывании макроса `\Pn` текст `$x\in\Pn$` превращается в незаконный текст `$x\in $\mathbf P^n$$`, в котором математическая формула заканчивается со вторым из знаков доллара, а символ `^` оказывается посреди обычного текста. Чтобы можно было напечатать $\mathbf{P}^n$ не только изолированно, не надо включать знаки доллара в определение:

```
\newcommand{\Pn}{\mathbf{P}^n}
```

При этом придется, конечно, ставить знаки доллара вокруг `\Pn` в тех случаях, когда в тексте встречается просто $\mathbf{P}^n$, но зато наш макрос можно будет использовать и как составную часть более сложных формул.

Есть, впрочем, и более удачный способ борьбы с этой проблемой: определите `\Pn` как

```
\newcommand{\Pn}{\ensuremath{\mathbf{P}^n}}
```

(без всяких знаков доллара) — и вы сможете спокойно пользоваться своей новой командой `\Pn` как в тексте, так и в формулах:

```
Пусть \Pn~--- $n$-мерное проективное пространство,
а $X\subset \Pn$~--- неприводимое многообразие...
```

(знаки `~` мы поставили, чтобы строчка не смогла начаться с тире — с. 103). Команда `\ensuremath` *всегда* обрабатывает свой аргумент как математическую формулу, независимо от того, в тексте или в формуле вы ее используете.

Создавать макросы полезно не только для сокращения числа нажатий на клавиши при наборе формул. Вот пример, когда макросы помогают и при наборе обычного текста. Предположим, в нашем тексте много задач, причем условие каждой из задач начинает новый абзац (как обычно и бывает). Предположим также (временно), что эти задачи никак не нумеруются. Слово «Задача», с которого начинается условие, хочется как-то выделить в тексте; предположим, мы решили выделять его жирным шрифтом. Давайте создадим макрос, который будет делать все это за нас, чтобы можно было не печатать каждый раз слово `Задача`, а просто написать `\z`. Первым обычно приходит в голову что-нибудь такое:

```
\newcommand{\z}{\bfseries Задача}
```

Посмотрите, что из этого выйдет:

Задача. Пять парней за пять дней съели пять окуней. За сколько дней пятнадцать парней съедят пятнадцать окуней?

```
\z. Пять парней за пять дней
съели пять окуней. За сколько
дней пятнадцать парней
съедят пятнадцать окуней?
```

Почему же жирным шрифтом напечаталось не только слово «Задача», но и весь дальнейший текст? Ответ: TeX опять пунктуально заменил `\z` на «замещающий текст», в результате чего получилось вот что:

```
\bfseries Задача. Пять парней ...
```

Команда `\bfseries` оказалась не внутри группы, и весь текст напечатался жирным шрифтом. Чтобы не попадаться в такую ловушку, надо помнить, что при развертывании макроса фигурные скобки, ограничивающие замещающий текст в команде `\newcommand`, отбрасываются. Правильнее было бы дать такое определение:

```
\newcommand{\z}{{\bfseries Задача}}
```

На сей раз `\z` будет заменяться на `{\bfseries Задача}`, чего мы и хотели (при развертывании макроса отбрасывается внешняя пара фигурных скобок, ограничивающая второй аргумент команды `\newcommand`, и только она!). А можно (это, пожалуй, даже лучше) написать и так:

```
\newcommand{\z}{\textbf{Задача}}
```

Впрочем, наш макрос `\z` еще не идеален. Во-первых, после слова, напечатанного жирным шрифтом, точку лучше поставить тоже жирным шрифтом, поэтому разумно и ее включить в макроопределение. Далее, согласно общему правилу игнорирования пробелов после имени команды, состоящего из букв, в тексте при этом нельзя будет написать

```
\z Пять парней...
```

так как при этом пропадет пробел между словом «**Задача.**» и следующим словом. Можно этот пробел, как водится, всякий раз специально организовывать и писать

```
\z{} Пять парней...
```

но лучше вставить и пробел прямо в определение:

```
\newcommand{\z}{\textbf{Задача. }}
```

Теперь запись `\z Пять парней...` даст то, что нужно. Впрочем, если угодно, вот еще один штрих. Если вы в какой-нибудь момент забудете оставить пустую строку перед очередной задачей, то слово «**Задача**» при этом будет не начинать абзац, а продолжать предшествующий текст. Чтобы этого раз и навсегда избежать, можно вставить команду «закончить абзац» в наше макроопределение. Как вы помните, эта команда называется `\par` (с. 142):

```
\newcommand{\z}{\par\textbf{Задача. }}
```

Если перед нашим макросом `\z` пустая строка все-таки будет, то лишняя команда `\par` ни на что не повлияет (см. с. 142 и ниже), а если ее не будет, то `\par` сыграет роль недостающей пустой строки.

Если какой-то элемент текста оформлен с помощью макроса, становится удобно менять это оформление. Например, если вдруг понадобилось оформлять все задачи так, чтобы соответствующие абзацы начинались без отступа, достаточно внести небольшое изменение в наше определение команды `\z`:

```
\newcommand{\z}{\par\noindent\textbf{Задача. }}
```

Если бы каждую задачу мы оформляли вручную, нам пришлось бы перед каждым словом «**Задача**» вписывать команду `\noindent`. В дальнейшем, познакомившись с LaTeX'овскими «счетчиками», мы усовершенствуем наш макрос `\z` таким образом, чтобы он еще и автоматически нумеровал задачи.

Мы уже говорили, что переопределить значение уже существующей команды с помощью `\newcommand` невозможно. Иногда, однако, это бывает необходимо. Пример тому приведен в гл. IV: если мы хотим, чтобы в заголовках глав печаталось именно слово «Глава», а не «Chapter», то необходимо определить по-новому команду `\chaptername`. Для такого рода целей используется команда `\renewcommand`. Она устроена точно так же, как `\newcommand`, с тем отличием, что в качестве ее первого аргумента надо указывать имя *уже существующей* команды; в этом случае по выполнении команды `\newcommand` значение этой команды изменится: она превратится в сокращенное обозначение для текста, указанного в качестве ее второго аргумента. Например, если написать

```
\renewcommand{\alpha}{Ку-ку}
```

то команда `\alpha` будет генерировать не то, что обычно (букву α в математической формуле и сообщение об ошибке, если эта команда употреблена вне математической формулы), а текст «Ку-ку».

Если команда `\renewcommand` дана внутри группы, то созданное ею переопределение значения команды забудется по выходе из этой группы. Если в качестве первого аргумента команды `\renewcommand` указать имя несуществующей команды, то вы получите сообщение об ошибке.

Команда `\renewcommand` при неаккуратном обращении может привести к неприятностям. Дело в том, что нередко команды LaTeX'а определяются в терминах других команд LaTeX'а, причем знать эти сложные связи рядовой пользователь вовсе не обязан. На практике такое незнание может привести к тому, что безобидное на первый взгляд использование в своих целях имени какой-нибудь команды, которая в вашем тексте ни разу не встречается, вызовет необъяснимо неправильную работу других команд. Поэтому используйте `\renewcommand` только тогда, когда вы полностью отдаете себе отчет в своих действиях.

Выше мы часто называли макросы просто командами, и это — не просто небрежность речи: на самом деле принципиальной разницы между макросами и TEX'овскими командами почти нет. В частности, почти все команды LATEX'а, о которых говорилось и еще будет говориться в нашей книге, являются именно макросами; в отличие от макросов, создаваемых нами с помощью `\newcommand`, их смысл уже известен TEX'у к моменту запуска программы, так что определять их каждый раз не нужно. Макросы, в свою очередь, могут ссылаться на другие макросы (кстати, команда `\stackrel`, на которую ссылалась наша команда `\eqdef`, также является макросом) — и так далее, пока не дойдет до так называемых «примитивных команд» TEX'а. Из команд, о которых мы вам рассказывали, примитивными являются считанные единицы: `\left`, `\right`, `\noindent` и некоторые другие.

1.2. Команды с аргументами

В предыдущем разделе мы научились создавать новые команды, не требующие аргументов. Но мы хорошо знаем, что многие LATEX'овские команды принимают аргументы, и часто возникает потребность создать новую команду с такими возможностями. Пусть, например, в вашем тексте часто встречается «символ Лежандра», выглядящий так: $\left(\frac{a}{l}\right)$. Для получения этого символа в исходном тексте надо написать (внутри формулы, естественно) так:

```
\left(\frac{a}{l}\right)
```

Хорошо бы создать команду `\smb` с двумя аргументами, чтобы можно было написать в формуле `\smb{a}{l}` и получить на печати $\left(\frac{a}{l}\right)$. Что ж, LATEX предоставляет нам возможность сделать и это. Для создания команды с аргументами используется все та же команда `\newcommand`, но с необязательным аргументом[2]. Посмотрите, как можно определить команду `\smb`:

```
\newcommand{\smb}[2]{\left(\frac{#1}{#2}\right)}
```

Разберем, что означает эта запись. В квадратных скобках стоит количество аргументов в нашем макросе (в нашем случае 2). Далее, в само́м «замещающем тексте» появились значки `#1` и `#2` . При развертывании макроса на их место будут подставляться соответственно первый и второй аргументы нашей новой команды `\smb`. Например, если в формуле написать

```
\smb{a+b}{c}
```

[2]Можно (и разумно) использовать в этой ситуации команду `\newcommand*` («вариант со звездочкой»). См. по этому поводу с. 242 ниже.

то будет напечатано $\left(\frac{a+b}{c}\right)$.

Теперь рассмотрим точные правила. Необязательный аргумент команды `\newcommand`, который должен быть расположен между двумя обязательными, указывает, сколько аргументов будет требовать создаваемая вами команда (макрос). Это количество аргументов не может быть более 9. В «замещающем тексте» места, на которые при развертывании макроса будут подставляться аргументы, обозначаются символами `#1` для первого аргумента, `#2` для второго аргумента и т. д. Эти символы могут идти в любом порядке и присутствовать любое количество раз (в том числе и ни разу). Когда мы будем использовать нашу новую команду в тексте, после ее имени в фигурных скобках должны будут следовать аргументы, ровно в том количестве и в том порядке, который мы указывали в необязательном аргументе команды `\newcommand`, каждый — в своей паре фигурных скобок (как обязательные аргументы любой другой LaTeX'овской команды). При развертывании макроса на место его и его аргументов будет подставлен «замещающий текст», в котором вместо `#1` всюду стоит первый аргумент, вместо `#2` — второй аргумент и т. д.

Проиллюстрируем все сказанное примером. Пусть мы определили команду `\shuffle` следующим образом:

```
\newcommand{\shuffle}[4]{К#4к#2#1л}
```

Тогда будет получаться, например, такое:

Крокодил Гена и его друзья.	`\shuffle{ди}{о}{го}{ро}` `Гена и его друзья.`

В самом деле, первым аргументом команды `\shuffle` будет `ди`, вторым `о`, третьим `го`, четвертым `ро`; при развертывании появится сначала буква `К`, затем четвертый аргумент, затем буква `к`, затем второй, первый и наконец буква `л` — как если бы слово `Крокодил` присутствовало в исходном тексте.

Ни аргументы LaTeX'овских команд (в том числе и определенных вами), ни «замещающий текст» в `\newcommand` не должны содержать «несбалансированных» (не имеющих пары) фигурных скобок (это не относится к `\{` и `\}`).

Если вы хотите создать макрос с аргументами, имя которого совпадает с именем уже существующей команды, то надо воспользоваться командой `\renewcommand` с необязательным аргументом. Место постановки и значение этого необязательного аргумента, а также правила употребления символов `#1`, `#2` и т. д. при этом будут такие же, как для команды `\newcommand`.

Приведем еще один пример практически полезного макроса с аргументами. При написании этой книги автор широко пользовался ссылками на страницу, автоматически генерируемыми с помощью команды `\pageref`. Например, если какое-то место в тексте было помечено с помощью команды `\label{units}`, то ссылка на соответствующую страницу выглядела так:

```
Как мы уже отмечали на с.~\pageref{units}, ...
```

После первого десятка таких ссылок возникает желание сократить число нажатий на клавиши. В результате в преамбуле появилась строка

```
\newcommand{\str}[1]{стр.~\pageref{#1}}
```

и ссылки на страницы стало возможно оформлять так:

```
Как мы уже отмечали
на \str{units}, ...
```

(Автор вначале не знал, что надо писать «с.», а не «стр.», но мгновенно исправился, всего лишь удалив две буквы из определения команды `\str`!)

2. Команда \newcommand со звездочкой

У команд `\newcommand` и `\renewcommand` существуют варианты «со звездочкой» (см. с. 25), называемые `\newcommand*` и `\renewcommand*`. Они работают точно так же, как их тезки без звездочек, со следующим отличием: если команда с аргументами определена с помощью `\newcommand*` или `\renewcommand*`, то в ее аргументе *не может содержаться пустая строка или команда* `\par`[3]. Если ваша команда определена с помощью `\newcommand` или `\renewcommand` без звездочки, то никаких ограничений на этот счет нет.

Смысл этого запрета таков. В большинстве случаев команды с аргументами, которые вы определяете, все равно не будут предусматривать подстановку вместо аргумента фрагмента текста, содержащего пустую строку или `\par`, так что этот запрет ни на чем не скажется (во всех примерах, приведенных в этой книге к настоящему моменту, можно совершенно безболезненно заменить `\newcommand` и `\renewcommand` на их варианты «со звездочкой»). А вот диагностика ошибок при наличии такого запрета облегчается. В самом деле, представим себе, что вы забыли набрать закрывающую фигурную скобку в аргументе команды

[3]На содержимое самого́ «замещающего текста» (второго обязательного аргумента `\newcommand*`) никаких дополнительных запретов не накладывается.

(весьма распространенная ошибка!), как в следующем примере (забыта фигурная скобка в самой первой формуле; команду `\smb` мы определили в учебных целях на с. 240):

```
Символ Лежандра $\smb{a{l}$, где $a$ не делится на~$l$,
равен по определению~$1$, если $a$ является квадратичным
вычетом по модулю~$l$, и~$-1$ в противном случае.

Вопрос о том, как зависит $\smb{a}{l}$ от $l$ при
фиксированном~$a$, является весьма важным и трудным.
Великий немецкий математик К.-Ф.\,Гаусс...
```

Если пустые строки в аргументе команды `\smb` не запрещены (так оно и будет, если определять `\smb` с помощью `\newcommand` без звездочки, как у нас на с. 240), то TeX будет терпеливо ждать, когда ему встретится закрывающая фигурная скобка, парная к первой из открывающих фигурных скобок в первой строке, а пока таковой нет — рассматривать весь читаемый им текст как составную часть первого аргумента команды `\smb`. В конце концов TeX либо доложит, что он прочел уже весь файл, а закрывающей фигурной скобки так и не нашел, либо прервет работу, объявив, что ему не хватило памяти.

Если же мы определим `\smb` с помощью `\newcommand*`, написав

```
\newcommand*{\smb}[2]{\left(\frac{#1}{#2}\right)}
```

то ошибка не пойдет дальше текущего абзаца: как только TeX увидит пустую строку среди текста, рассматриваемого им как аргумент команды, он тут же прервет дальнейшее чтение и выдаст следующее стандартное сообщение об ошибке:

```
Runaway argument?
{a{l}$, где $a$ не делится на~$l$, равен п\ETC.
! Paragraph ended before \smb was complete.
<to be read again>
                   \par
l.8

?
```

Нажав пару раз на «ввод», мы сможем благополучно продолжить обработку текста. В первом абзаце пропадет все после слов «Символ Лежандра», зато второй и последующие абзацы будут обработаны нормально (пока TeX не наткнется на очередную ошибку...).

Команды `\newcommand*` и `\renewcommand*` обладают еще одним преимуществом перед своими вариантами без звездочек: при их использовании происходит (небольшая) экономия памяти.

Рекомендуем вам определять и переопределять команды с аргументами при помощи `\newcommand*` и `\renewcommand*`. Варианты без звездочек используйте только тогда, когда вы действительно намерены подставлять в аргумент своего макроса текст, состоящий из нескольких абзацев.

3. Счетчики

В этом разделе мы научимся самостоятельно организовывать автоматическую нумерацию, подобно тому, как LaTeX автоматически нумерует главы, страницы, формулы и т. д. Для этого нам надо познакомиться с понятием счетчика. Счетчик — это специальная переменная, значение которой является целым числом. Ей можно присваивать различные значения, выводить значение счетчика на печать, а также организовывать с помощью счетчиков автоматическую генерацию ссылок. Рассмотрим последовательно, как все это делается.

3.1. Создание счетчиков и простейшие операции с ними

Каждый LaTeX'овский счетчик имеет свое имя — последовательность букв (без знака `\`). Прописные и строчные буквы в именах счетчиков различаются; если в используемой вами русификации русские буквы нельзя употреблять в именах команд, то нельзя их употреблять и в именах счетчиков. Чтобы можно было работать со счетчиком, надо его создать командой `\newcounter`, имеющей один обязательный аргумент — имя, которое вы придумали для счетчика. Например, команда

```
\newcounter{abcd}
```

создает новый счетчик с именем `abcd`. Если имя, которое вы придумали для счетчика, уже занято (так может случиться, даже если команда `\newcounter` в вашем тексте всего одна: некоторые счетчики в LaTeX'е уже определены в момент начала обработки вашего текста), то LaTeX создавать счетчик с таким именем откажется и выдаст сообщение об ошибке.

В отличие от многих других LaTeX'овских команд, команда для создания нового счетчика является «глобальной»: даже если она давалась внутри группы, LaTeX не забудет о существовании определенного ею счетчика и после выхода из этой группы. Это отличает `\newcounter` от `\newcommand` и `\renewcommand`.

Что же можно делать со счетчиком? Во-первых, можно менять его численное значение (на программистском жаргоне: «присваивать счетчику различные значения»). При создании счетчика его значение устанавливается в 0; чтобы установить какое-то другое значение, используется команда `\setcounter`, имеющая два обязательных аргумента: первый — имя счетчика, второй — значение, которое счетчику присваивается. Если, например, написать

```
\setcounter{abcd}{1998}
```

то после того, как TeX прочтет эту команду, значение счетчика `abcd` установится равным 1998. Значение, которое присваивается счетчику, может быть и отрицательным, но обязано быть целым.

Другой способ изменить значение счетчика — это прибавить к нему какое-то целое число. Для этого используется команда `\addtocounter`. Эта команда также имеет два обязательных аргумента: первый — имя счетчика, второй — число, которое прибавляется к счетчику. Например, после выполнения команд

```
\setcounter{abcd}{100}
\addtocounter{abcd}{-27}
```

значением счетчика `abcd` будет число 73.

Команды, изменяющие значение счетчика, также являются «глобальными»: если с их помощью внутри группы значение счетчика было изменено, то по выходе из группы его прежнее значение не восстановится.

Перейдем к самому главному: как вывести значение счетчика на печать. Самый распространенный случай — печать значения счетчика обычными («арабскими») цифрами. Для этого используется команда `\arabic`:

Шел по улице отряд — 40 мальчиков подряд.

```
\setcounter{abcd}{40}
Шел по улице отряд~---
\arabic{abcd}
мальчиков подряд.
```

Значение счетчика печатается текущим шрифтом: если значение счетчика равно, скажем, 2003, то на команду `\arabic{abcd}` TeX отреагирует так же, как если бы на ее месте в исходном тексте было написано `2003`.

Чтобы напечатать значение счетчика римскими цифрами, надо воспользоваться командой `\Roman` (если мы хотим, чтобы римские цифры записывались прописными латинскими буквами) или `\roman` (чтобы записать римскую цифру строчными латинскими буквами):

Людовика XIV звали “Король-Солнце”.

```
\setcounter{abcd}{14}
Людовика~\Roman{abcd} звали
``Король-Солнце''.
```

Естественно, при печати значения счетчика римскими цифрами это значение должно быть положительным числом.

Можно, наконец, напечатать букву латинского алфавита, порядковый номер которой равен значению счетчика. Для этого используются команды `\alph` (для печати строчной буквы) и `\Alph` (для печати прописной буквы):

Наконец я узнаю, какая буква стоит в латинском алфавите на седьмом месте! Вот она: g.

```
\setcounter{abcd}{7}
Наконец я узнаю, какая буква
стоит в латинском алфавите
на седьмом месте!
Вот она:~\alph{abcd}.
```

Если значение счетчика при пользовании этими командами превышает количество букв в латинском алфавите, то LaTeX выдает сообщение об ошибке.

В LaTeX'е отсутствует команда, позволяющая напечатать *русскую* букву с номером, равным значению счетчика. Средствами TeX'а такую команду нетрудно создать; см. разд. IX.2.3 по поводу того, как это сделать.

Наконец, можно напечатать один из девяти символов, используемых иногда в англоязычных странах для обозначения последовательных сносок (вместо цифр). Для этого используется команда `\fnsymbol`, применять которую можно только внутри формул:

Для сносок в Англии применяют такие символы: $*$, $\dagger$, $\ddagger$, а дальше попробуйте сами.

```
\setcounter{abcd}{0}
Для сносок в Англии
применяют такие символы:
$\addtocounter{abcd}{1}\fnsymbol{abcd}$,
$\addtocounter{abcd}{1}\fnsymbol{abcd}$,
$\addtocounter{abcd}{1}\fnsymbol{abcd}$,
а дальше попробуйте сами.
```

Обратите внимание, как три идентичных фрагмента исходного текста дали на печати три разных символа.

В командах для изменения значений счетчиков можно вместо явного указания чисел использовать значения других счетчиков, для чего употребляется команда `\value`. Например, в результате выполнения команд

```
\newcounter{efgh}
\setcounter{abcd}{10}
\setcounter{efgh}{100}
\addtocounter{efgh}{-\value{abcd}}
```

значение счетчика `efgh` станет равно 90. Можно даже писать, например,

```
\setcounter{efgh}{1000}
\tolerance=\value{efgh}
```

но большого смысла в таких трюках, как правило, нет.

Применим наши познания к делу. На с. 239 мы обещали вам так усовершенствовать макрос `\z`, начинающий новый абзац и печатающий жирным шрифтом слово «**Задача**», чтоб он еще и автоматически нумеровал эти задачи, так, что можно было бы просто писать в исходном тексте

```
\z Найти сумму...
\z Решить уравнение...
\z Поезд вышел из пункта A...
```

и при этом знать, что номера LaTeX проставит сам. Теперь мы в состоянии решить эту проблему. Во-первых, для этого надо создать счетчик, значение которого в каждый момент будет равно номеру последней обработанной задачи; во-вторых, в определении команды `\z` надо предусмотреть, чтобы всякий раз значение этого счетчика увеличивалось на единицу, а затем печаталось в качестве номера задачи. В качестве имени счетчика выберем бесхитростное «zadacha»:

```
\newcounter{zadacha}
```

(напомним, что при выполнении этой команды счетчику zadacha будет присвоено значение 0). Теперь модифицируем определение макроса `\z` так:

```
\newcommand{\z}{\par\addtocounter{zadacha}{1}%
\textbf{Задача \arabic{zadacha}.} }
```

Напомним, что команда `\par` означает «завершить предыдущий абзац, если он еще не был завершен»; без нее можно обойтись, если мы будем ставить команду `\z` только после пустой строки. Знак процента мы поставили, чтобы убрать лишний пробел, порождаемый концом строки. Теперь при первом исполнении команды `\z` значение счетчика zadacha станет равно 1 и будет напечатано «**Задача 1.**», при втором исполнении этой команды значение счетчика станет равно уже 2 и напечатается «**Задача 2.**» … и т. д., что нам и нужно!

3.2. Отношение подчинения между счетчиками

Команда `\z`, как мы ее определили в предыдущем разделе, нумерует задачи автоматически, но при этом нумерация получается «сплошной». Часто, однако, требуется, чтобы в каждом разделе документа нумерация задач начиналась заново, так что шестая задача в разделе с номером 3 была озаглавлена **Задача 3.6**, а первая задача в разделе с номером 4 — **Задача 4.1**. Сейчас мы узнаем, как этого добиться.

Выше мы упоминали, что к моменту начала обработки LaTeX'ом нашего текста некоторые счетчики уже определены. В частности, это счетчики, содержащие номера текущих разделов документа. Их имена совпадают с именами команд, генерирующих эти разделы: `chapter` (если классом предусмотрено разбиение на главы), `section`, `subsection` и т. д. При каждом исполнении, например, команды `\section` значение счетчика `section` увеличивается на 1, и значение этого счетчика в каждый момент равно номеру текущего раздела. Поэтому, если в определении команды `\z` написать

```
\arabic{section}.\arabic{zadacha}.
```

то перед номером задачи будет печататься номер текущего раздела и точка.

Но как же все-таки сделать, чтобы в каждом разделе нумерация задач начиналась заново? Можно, конечно, в начале каждого раздела присваивать счетчику `zadacha` значение 0 с помощью `\setcounter`, но это некрасиво и ненадежно (а вдруг забудем?). Лучше сразу определить счетчик `zadacha` так:

```
\newcounter{zadacha}[section]
```

При этом счетчик `zadacha` будет *подчинен* счетчику `section`: всякий раз, когда значение счетчика `section` увеличивается на единицу командой `\section`, значение счетчика `zadacha` будет устанавливаться в нуль, и тем самым счет задач будет в каждом разделе начинаться заново. Одновременно надо в очередной раз исправить определение команды `\z` и написать

```
\newcommand{\z}{\par\addtocounter{zadacha}{1}%
\textbf{Задача \arabic{section}.\arabic{zadacha}.} }
```

При этом нумерация задач будет начинаться заново в каждом разделе, и вторая задача третьего раздела будет иметь номер 3.2.

Теперь сообщим точные правила создания счетчика, подчиненного другому счетчику. Они просты: команда `\newcounter` может принимать один необязательный аргумент (*после* обязательного) — имя того счетчика, которому будет подчинен определяемый нами счетчик. Разумеется, в момент выполнения команды `\newcounter` с необязательным аргументом счетчик, имя которого дается в квадратных скобках, должен уже существовать.

Надо еще уточненить, в каких случаях значение подчиненного счетчика устанавливается в нуль. В самом деле, пусть счетчик `slave` подчинен счетчику `master`; тогда команда

```
\addtocounter{master}{1}
```

никоим образом не повлияет на значение подчиненного счетчика `slave`: изменение значений счетчика влияет на значения подчиненных ему счетчиков только в том случае, если значение подчиняющего счетчика изменялось с помощью специальных команд. Таких команд всего две, из них чаще всего используется `\refstepcounter`: она увеличивает на единицу значение указанного ей счетчика, а значения всех подчиненных ему счетчиков устанавливает в нуль. Пусть, например, в нашем тексте определены два счетчика:

```
\newcounter{master}
\newcounter{slave}[master]
```

Тогда после выполнения команд

```
\setcounter{master}{10}
\setcounter{slave}{10}
```

значения обоих счетчиков станут равны 10, после выполнения команды

```
\addtocounter{master}{1}
```

значение счетчика `master` станет равно 11 и значение счетчика `slave` не изменится, а вот после выполнения команды

```
\refstepcounter{master}
```

значение счетчика `master` станет равно 12, в то время как значение счетчика `slave` станет равно нулю.

Наряду с `\refstepcounter` существует еще одна команда, изменяющая значение счетчика таким образом, что значения всех подчиненных ему счетчиков устанавливаются в нуль. Эта команда называется `\stepcounter`; она также увеличивает на единицу значение счетчика, имя которого является ее аргументом, и при этом обнуляет все подчиненные ему счетчики, но она непригодна для организации автоматических ссылок (см. следующий раздел), вследствие чего область ее применения более ограничена.

Хороший пример использования подчиненных счетчиков дают стандартные LaTeX'овские классы документов. Например, в классе `book` перед началом обработки текста выполняются следующие команды:

```
\newcounter{part}
\newcounter{chapter}
\newcounter{section}[chapter]
\newcounter{subsection}[section]
\newcounter{subsubsection}[subsection]
\newcounter{paragraph}[subsubsection]
\newcounter{subparagraph}[paragraph]
```

Стало быть, нумерация глав не зависит от нумерации частей (если третья часть книги завершается десятой главой, то четвертая часть начинается с одиннадцатой главы), а нумерация разделов уже начинается заново в каждой главе.

3.3. Организация автоматических ссылок

Вернемся последний раз к нашей команде `\z`. Раз она автоматически нумерует задачи, то неплохо было бы, если б пронумерованные ею задачи можно было метить командой `\label` и ссылаться на эти метки командой `\ref` (проблема именно в ней, поскольку команда `\pageref`, дающая номер страницы, сработает в любом случае). Если коротко, то решение этой проблемы таково: увеличивать на единицу значение счетчика `zadacha` надо не с помощью команды `\addtocounter`, которой мы пользовались до сих пор, а с помощью команды `\refstepcounter`, о которой уже шла речь по другому поводу в предыдущем разделе. Если мы определим команду `\z` так:

```
\newcommand{\z}{\par\refstepcounter{zadacha}%
\textbf{Задача \arabic{section}.\arabic{zadacha}.} }
```

то после этого можно будет написать, например, так:

```
\z Решить уравнение...
\z Доказать...\label{prove}
\z Найти сумму...
```

Если теперь в другом месте текста мы сошлемся на помеченную задачу так:

```
В задаче \ref{prove} предлагалось доказать...
```

то будет печататься ее номер (тот самый, который LaTeX автоматически ей присвоил). Впрочем, с такими автоматическими ссылками не все будет благополучно: если помеченная нами задача была второй по счету в разделе номер 3, то называться она будет **Задача 3.2**, а вот ссылка на нее, сгенерированная командой `\ref`, будет выглядеть просто

В задаче 2 предлагалось доказать...

в то время как хотелось бы автоматически получить «В задаче 3.2». Иными словами, надо изменить текст, генерируемый командой `\ref`. Чтобы узнать, как этого добиться, нам придется познакомиться с еще одной LaTeX'овской конструкцией, связанной со счетчиками.

Мы уже знаем, что значение LaTeX'овского счетчика можно вывести на печать командами `\arabic`, `\roman` и т. п. Однако, кроме этого, с каждым счетчиком связана индивидуальная команда, определяющая, в какой форме его значение будет выводиться на печать, и именно в соответствии с этой командой печатается ссылка, сгенерированная с помощью `\label` и `\ref`. Имя этой команды получается, если поставить `the` перед именем счетчика. Например, команда для вывода на печать номера раздела называется `\thesection`, для вывода на печать номера главы — `\thechapter`, команды для вывода на печать определенных нами счетчиков `slave` и `master` — `\theslave` и `\themaster`. При создании счетчика автоматически определяется и соответствующая `the`-команда. Именно, при создании счетчика по имени, скажем, `abcd` автоматически определяется команда `\theabcd` таким образом:

```
\newcommand{\theabcd}{\arabic{abcd}}
```

В дальнейшем эту команду можно переопределять:

Людовика XIV звали “Король-Солнце”.

```
\renewcommand{\theabcd}%
{\Roman{abcd}}
\setcounter{abcd}{14}
Людовика~\theabcd{} звали
``Король-Солнце''.
```

Мы же, чтобы при ссылках перед номером задачи печатался номер раздела, в котором находится эта задача, и точка, переопределим команду `\thezadacha` так:

```
\renewcommand{\thezadacha}{\thesection.\arabic{zadacha}}
```

Если включить эту команду в преамбулу документа, то ссылки на сгенерированные нашей командой `\z` номера задач будут выглядеть должным образом.

В нашем переопределении команды `\thezadacha` мы воспользовались командой `\thesection`, чтобы наши макросы правильно работали с любым классом документов. Дело в том, что при разумном оформлении номер раздела, предшествующий при ссылке номеру задачи, должен печататься таким же образом, как и номер раздела при ссылке на раздел, а это в разных классах делается по-разному: в классе `article`, например, `\thesection` — это то же самое, что и `\arabic{section}` (иными словами, ссылка на раздел, сгенерированная командой `\ref`, печатает просто номер раздела), а в классе `report` команда `\ref` при печати ссылки на раздел печатает не номер раздела, а номер главы, точку и номер раздела. Поскольку мы написали `\thesection`, все эти тонкости будут учтены автоматически.

Приведем еще один (игрушечный) пример использования счетчиков в макроопределениях, отчасти чтобы еще раз продемонстрировать применение подчиненных счетчиков, а отчасти чтобы убедить читателя, что не боги горшки обжигают. Именно, давайте разработаем свои собственные команды для создания разделов документа, не полагаясь на `\chapter`, `\section` и т. п. из стандартных LaTeX'овских классов. Для простоты предположим, что заголовки глав и разделов будут укладываться в одну строку, и не будем заботиться о том, насколько удачным выйдет оформление заголовков. Итак, пусть наш документ делится на главы, которые, в свою очередь, делятся на разделы. Каждую главу будем начинать с новой страницы, перед каждым разделом оставлять `1cm` (если, конечно, раздел не начинает новой страницы). Наконец, предусмотрим, чтобы главы и разделы автоматически нумеровались с возможностью создания автоматических ссылок на эти номера. Команды для создания главы и раздела назовем `\glava` и `\razdel`; это будут команды, требующие одного аргумента — названия главы или раздела.

Чтобы номера глав и разделов генерировались автоматически, нам необходимо создать счетчики, содержащие эти номера. Пусть счетчик с номером главы называется `glava`, а счетчик с номером раздела — `razdel` (имена счетчиков могут совпадать с именами команд). Имея в виду, что нумерация разделов в каждой главе будет начинаться заново, напишем в преамбуле так:

```
\newcounter{glava}
\newcounter{razdel}[glava]
```

Теперь определим команду `\glava`:

```
\newcommand*{\glava}[1]{\clearpage % с новой страницы
\vspace*{4cm}% оставить место сверху
\refstepcounter{glava}% новый номер главы
{\LARGE\rmfamily\bfseries\upshape % шрифт для заголовка
Глава \theglava. #1% заголовок
\par % кончить заголовок
\vspace{5mm plus 1mm minus .5mm}% Промежуток между
% заголовком и текстом
}% завершить группу, внутри которой менялся шрифт
}% конец макроопределения
```

Поскольку номер главы устанавливается командой `\refstepcounter`, при этом будет начата заново и нумерация разделов, а на номера глав можно будет делать автоматические ссылки с помощью `\label` и `\ref`. В определении шрифта для заголовков мы в явном виде установили все четыре атрибута, чтобы быть уверенными, что он будет какой надо (если бы мы, например, не сказали `\upshape`, а предыдущая глава заканчивалась курсивом, определенным командой `\itshape`, данной вне группы, то заголовок напечатался бы жирным курсивом).

Обратите также внимание на команду `\theglava`. Мы воспользовались ею, поскольку не хотим на этом этапе предрешать, в каком виде номер главы будет представлен в заголовке: в виде арабской цифры, римской цифры или еще как-нибудь. Если в дальнейшем нам захочется изменить вид этого представления, то не придется менять, с риском ошибиться, длинное определение команды `\glava`, а будет достаточно переопределить команду `\theglava`. Заметьте, что промежуток между заголовком и текстом мы сделали растяжимым, чтобы помочь TEX'у найти правильное разбиение на страницы (см. с. 147).

Определение команды `\razdel` можно дать, например, так:

```
\newcommand{\razdel}[1]{\par % завершить предыдущий абзац
\pagebreak[2]\vspace{1cm plus 3mm minus .5mm} % см. ниже
\refstepcounter{razdel}% новый номер раздела
{\Large\rmfamily\bfseries\upshape % шрифт для заголовка
\therazdel{} #1% заголовок
\par % закончить заголовок
}% завершить группу, внутри которой менялся шрифт
\nopagebreak % чтобы не оторвать текст от
             % заголовка
\vspace{2mm plus 1mm}% Промежуток между заголовком
                     % и текстом
}% конец макроопределения
```

Пояснений тут требует команда `\pagebreak[2]`. Мы включили ее в макроопределение, чтобы поменьше разделов начиналось внизу страницы. В самом деле, команда `\pagebreak[2]` предлагает TEX'у начать в этом месте новую страницу (см. раздел III.9.3); если так и будет сделано, то дополнительный вертикальный промежуток, созданный командой `\vspace`, пропадет, и заголовок раздела начнется с самого верха новой страницы; если же разрыва страницы все-таки не произойдет, то перед заголовком раздела будет вертикальный промежуток в один сантиметр (обладающий указанными в макроопределении растяжимостью и сжимаемостью).

Нам осталось только задать вид, в котором будут представляться на печати номера глав и разделов. Иными словами, надо переопределить должным образом команды `\theglava` и `\therazdel` (в момент создания счетчиков они, как мы помним, были автоматически определены таким образом, что `\theglava` и `\therazdel` — номера главы и раздела, набранные арабскими цифрами). Предположим, мы решили, что номера глав будут печататься римскими цифрами, а номер второго раздела четвертой главы будет иметь вид IV–2. Тогда требуемые переопределения таковы:

```
\renewcommand{\theglava}{\Roman{glava}}
\renewcommand{\therazdel}{\theglava--\arabic{razdel}}
```

Еще одно замечание: точки после номера главы мы включили в определение команды `\glava`, а не `\theglava`, чтобы можно было пользоваться автоматическими ссылками: если бы `\theglava` определялось как

```
\Roman{glava}.
```

то исходный текст

```
в главе \ref{metka} мы пишем...
```

дал бы на печати

в главе IV. мы пишем

что нелепо.

По сравнению с макроопределениями, реально используемыми в стандартных классах LaTeX'а, мы многого не предусмотрели: не позаботились ни о колонтитулах, ни об автоматически генерируемом оглавлении, внешний вид заголовков оставляет желать лучшего и т. д. Прочитав главу IX, вы сможете усовершенствовать эти определения.

3.4. Счетчики, которые уже определены

Мы уже мельком упоминали, что при начале работы LaTeX'а некоторые счетчики определены сразу. Например, это те счетчики, которые перечислены на с. 249. Кроме того, заранее определен счетчик `page`, отвечающий за нумерацию страниц, а также счетчик `footnote`, ответственный за нумерацию сносок. Нумерацией плавающих иллюстраций и таблиц занимаются счетчики, называемые `figure` и `table`. В разд. IX.2 перечислены все эти заранее определенные счетчики и указано, каким счетчикам они подчинены (это иногда зависит от класса документа).

Для каждого из этих счетчиков вы имеете возможность переопределить соответствующую `the`-команду и тем самым изменить стиль оформления документа. Например, вы можете сделать так, чтобы главы нумеровались римскими цифрами:

```
\renewcommand{\thechapter}{\Roman{chapter}}
```

Если вы хотите, чтоб сноски нумеровались не цифрами, а латинскими буквами, то можно в преамбуле написать:

```
\renewcommand{\thefootnote}{\alph{footnote}}
```

3.5. Модификация оформления перечней

Хороший пример переопределения команд доставляют перечни. В свое время (разд. III.7) мы обещали рассказать о том, как менять оформление перечней, задаваемых окружениями `itemize` и `enumerate`; сейчас мы, наконец, можем это сделать.

Начнем с `itemize`. Чтобы поменять значки, которыми помечаются элементы перечня, надо переопределить команду `\labelitemi`. Если, например, мы хотим, чтобы элементы перечня отмечались не черными кружками, а галочками $\surd$, то достаточно написать в преамбуле

```
\renewcommand{\labelitemi}{$\surd$}
```

(команду `\surd` см. в таблице на с. 53). Если окружение `itemize` расположено внутри другого окружения `itemize`, как в примере на с. 131, то значки для пометки элементов перечня будут уже, вообще говоря, другими: их вид задается командой `\labelitemii`; вид значков для пометок элементов `itemize` на третьем и четвертом уровнях вложенности задается командами `\labelitemiii` и `\labelitemiv`; их также можно переопределять.

Правильнее было бы определять заголовки для `itemize` чуть хитрее. Например, наше определение `\labelitemi` лучше дать так:

```
\renewcommand{\labelitemi}{$\mathsurround=0pt \surd$}
```

Если дать определение именно так, то вокруг галочки не появится дополнительный пробел даже в случае, если вы в какой-то момент решите установить ненулевое значение параметра `\mathsurround`. Поскольку формула образует группу, в дальнейшем предыдущее значение `\mathsurround` восстановится. Полезно иметь в виду этот прием, если вы пользуетесь математическими символами в качестве типографских значков.

Теперь рассмотрим окружение `enumerate`. Коль скоро оно автоматически нумерует элементы перечня, можно предположить, что это окружение связано с LaTeX'овскими счетчиками. Так оно на самом деле и есть: это окружение использует счетчик `enumi`. Если одно `enumerate` вложено в другое, то используются счетчики `enumii`, `enumiii` и `enumiv` для нумерации элементов перечня на втором, третьем и четвертом уровнях вложенности. С другой стороны, сами значки, помечающие элементы перечня, порождаются командами `\labelenumi`, `\labelenumii`, `\labelenumiii` и `\labelenumiv` — в зависимости от уровня вложенности. Например, команда `\labelenumi` определена так:

```
\newcommand{\labelenumi}{\theenumi.}
```

в то время как `the`-команда, определяющая представление счетчика `enumi` на печати, определена просто как

```
\newcommand{\theenumi}{\arabic{enumi}}
```

Стало быть, если мы не меняем стандартного стиля оформления, то элементы перечня `enumerate` (не вложенного в другой `enumerate`) будут нумероваться цифрами с точкой. Если же мы хотим, скажем, чтобы после цифры шла не точка, а скобка (как в нашей книге), то можно в преамбуле написать

```
\renewcommand{\labelenumi}{\theenumi)}
```

Если же мы к тому же хотим, чтобы элементы перечня нумеровались римскими цифрами, то можно написать еще и так:

```
\renewcommand{\theenumi}{\Roman{enumi}}
```

Аналогичным образом можно менять оформление нумерованных перечней на других уровнях вложенности.

Если вы хотите изменить оформление перечня более серьезным образом (например, установить другую величину полей, или сделать так, чтобы значки, помечающие элементы перечня, были выровнены по левому, а не по правому краю), то вам придется подождать до гл. IX.

4. Параметры со значением длины

Наряду со счетчиками — переменными с целочисленными значениями, — при создании собственных макроопределений возникает нужда и в переменных, значениями которых являются длины. Например, в предыдущем разделе мы, разрабатывая команду `\razdel`, в явном виде задали промежуток между заголовком раздела и остальным текстом. Если этот промежуток нам почему-либо захочется изменить, то придется снова залезать в определение команды `\razdel`. Было бы удобнее, если бы в нашем распоряжении был параметр под названием, скажем, `\otstup`, так что можно было бы в определении команды `\razdel` написать

```
\vspace{\otstup}
```

и потом отдельно написать, допустим,

```
\otstup=1cm plus 3mm minus .5mm
```

Правда, в богатом наборе TeX'овских и LaTeX'овских параметров требуемого нам параметра `\otstup` нет, но это не страшно, поскольку его можно создать. Для этого используется команда `\newlength`:

```
\newlength{\otstup}
```

После того, как вы (допустим, в преамбуле) дали эту команду, будет определен новый параметр со значением длины; его можно будет обычным образом использовать в аргументах команд наподобие `\vspace` и ему можно будет обычным образом присваивать значения.

Теперь — точные правила. Команда `\newlength` имеет один обязательный аргумент — имя команды, обозначающей определяемый вами параметр. Это имя должно подчиняться обычным правилам для TEX'овских команд (backslash, после которого следует либо одна небуква, либо последовательность букв). Если это имя уже занято, LATEX выдаст сообщение об ошибке. Определение нового параметра, совершаемое командой `\newlength`, является «глобальным»: даже если эта команда была дана внутри группы, TEX будет помнить о существовании этого параметра и по выходе из группы. По этой причине разумное место для команды `\newlength` — преамбула.

Определенный нами параметр со значением длины приобретает такой же статус, как уже существующие TEX'овские и LATEX'овские параметры (`\parindent`, `\textwidth` и другие). Рассмотрим, что можно делать с этими параметрами.

Во-первых, параметрам со значением длины можно присваивать значения. Делается это точно так же, как это объяснялось в разд. I.2.9 на примере параметра `\parindent`: для присваивания значения надо написать имя параметра, знак равенства, а после знака равенства — величину присваиваемой длины. Пробелы после указания единицы длины TEX'ом игнорируются (скорее всего, вы будете присваивать значения параметрам в преамбуле документа или между абзацами, где лишние пробелы никого не волнуют). Длина должна быть выражена в единицах, воспринимаемых TEX'ом (см. их список в разд. I.2.10). Даже если вы присваиваете нулевую длину, какая-то единица длины должна быть явно указана (например, `0pt`). Кроме того, можно воспользоваться LATEX'овской командой `\setlength`, имеющей два обязательных аргумента: первый — имя параметра, второй — значение длины, присваиваемое этому параметру. Таким образом, команды

```
\parindent=1.5em
```

и

```
\setlength{\parindent}{1.5em}
```

равносильны. Наконец, присваивания, сделанные внутри группы, забываются по выходе из этой группы.

В предыдущем абзаце мы умолчали об одной возможной неприятности. Дело в том, что если после команды присваивания, не использующей `\setlength`, следует (пусть даже после пробела) слово `plus` или `minus`, то TEX, скорее всего, выдаст сообщение об ошибке, поскольку решит, что длина должна иметь, помимо «естественного размера», еще и plus- или minus-компоненту (см. с. 147; ниже мы поговорим подробнее о такой возможности). Если вы пишете текст на русском языке, вероятность такого стечения обстоятельств ничтожна, но тем не менее забывать о такой опасности не следует, особенно если команда присваивания входит в макроопределение: вы же не знаете заранее, в какое место может попасть новый макрос. Чтобы застраховаться от этой неприятности раз и навсегда, пользуйтесь командой \setlength, хоть это и длиннее. Ср. также обсуждение команд \hrule и \vrule в разд. III.10.

Параметры со значением длины можно использовать всюду, где в аргументе LaTeX'овской команды требуется указать размер. Пусть, например, в преамбуле документа написано

```
\newlength{\primer}
```

Тогда посмотрите на следующий пример:

9 9 8 8 9 9	`\primer=10mm` `9\hspace{\primer}9` `{\primer=20mm` `8\hspace{\primer}8}` `9\hspace{\primer}9`

Обратите внимание, что, если присваивание параметру нового значения происходило внутри группы, то по выходе из группы новое значение забывается, а прежнее — восстанавливается.

Параметры со значением длины можно указывать с коэффициентом — положительной или отрицательной десятичной дробью (можно использовать как десятичную точку, так и десятичную запятую). Например, если значение параметра \primer равно 10 мм, то команда \hspace{2.71\primer} сделает пробел длиной 27.1 мм.

Параметры со значением длины (возможно, с числовыми коэффициентами) могут также стоять в правой части оператора присваивания (или во втором аргументе команды \setlength):

```
\primer=1.45\parindent
\setlength{\primer}{.45\tabcolsep}
```

Можно также прибавлять длину к значению параметра: если значение параметра \abcd равно x, то после выполнения команды

\addtolength{\abcd}{y}

где y — длина, значение параметра `\abcd` станет равно $x + y$. В качестве y в этой команде может использоваться как явно указанная длина (например, `1.2in`), так и параметр со значением длины (возможно, с числовым коэффициентом). Наконец, LaTeX предоставляет полезную команду

`\settowidth{`*параметр*`}{`*текст*`}`

которая присваивает *параметру* значение, равное ширине *текста*. Вот пример:

СЛОВО слово слово	`\settowidth{\primer}{\Large слово }` `{\Large слово }слово` `\hspace{\primer}слово`

(обратите внимание, что у нас получилось выравнивание без помощи `tabbing` или `tabular`).

Существуют также команды `\settoheight` и `\settodepth`, аналогичные `\settowidth`. Команда `\settoheight` присваивает *параметру* значение, равное максимальному расстоянию, на которое *текст* возвышается над строкой (точнее, над ее базисной линией — разд. VIII.1). `\settodepth` присваивает *параметру* значение, равное максимальному расстоянию, на которое *текст* опускается ниже базисной линии.

В разд. III.9.4 у нас шла речь о том, что некоторые используемые в TeX'е дли́ны могут обладать растяжимостью или сжимаемостью. Параметрам, созданным с помощью команды `\newlength`, также можно присваивать значения, содержащие `plus`- и/или `minus`-компоненту. Если, например, мы хотим, чтобы параметр `\primer` имел естественный размер 2 см, растяжимость 4 мм и сжимаемость в один пункт, то можно написать так:

```
\setlength{\primer}{2cm plus 4mm minus 1pt}
```

5. Создание новых окружений

5.1. Новые окружения: общий случай

Как мы уже имели возможность убедиться, для сокращения времени на написание длинных последовательностей команд удобно пользоваться макросами. В тех случаях, когда для достижения необходимого нам эффекта требуется сложная последовательность команд в начале и в конце какого-то текста, LaTeX дает возможность оформить соответствующие макросы в виде нового окружения. Как это делается, разберем на примере.

Предположим, нам хочется взять в рамку абзац текста шириной 7 см. Один из возможных способов таков:

```
\begin{tabular}{|p{7cm}|}
\hline
Этот текст будет заключен в рамку. Как видите,
окружение, предназначенное для верстки таблиц,
можно использовать и для этих целей.\\
\hline
\end{tabular}
```

При этом будет напечатано вот что:

Этот текст будет заключен в рамку. Как видите, окружение, предназначенное для верстки таблиц, можно использовать и для этих целей.

Если таких рамок с текстом у вас много, то можно сократить число нажатий на клавиши, определив окружение с именем, скажем, `ramka`, так, чтоб можно было бы просто писать

```
\begin{ramka}
Этот текст будет ...
... этих целей.
\end{ramka}
```

Определяется это окружение так:

```
\newenvironment{ramka}{\begin{tabular}{|p{7cm}|}
                       \hline}{\\\hline\end{tabular}}
```

В общем случае команда `\newenvironment` имеет такой формат:

`\newenvironment{`*имя*`}{`*открывающие_команды*`}{`*закрывающие_команды*`}`

Здесь *имя* — имя определяемого окружения, *открывающие_команды* — команды и/или текст, подставляемые вместо команды `\begin` с именем окружения, *закрывающие_команды* — команды и/или текст, подставляемые вместо команды `\end` с именем окружения.

Вместо окружения, определяемого с помощью `\newenvironment`, можно с тем же успехом создать два макроса: один — для *открывающих_команд*, другой — для *закрывающих*. Например, в нашем случае с рамкой можно было бы написать

```
\newcommand{\nachalo}{\begin{tabular}{|p{7cm}|}\hline}
\newcommand{\konec}{\\\hline\end{tabular}}
```

и создавать рамки так:

```
\nachalo
Этот текст...
\konec
```

Преимущество оформления такого рода конструкций в виде окружений состоит в том, что при этом легче контролировать ошибки: если вы напишете `\begin{ramka}` и при этом забудете написать соответствующую команду `\end{ramka}`, то LaTeX выдаст сообщение об ошибке, в котором именно это вам и скажет; если же вы забудете команду `\konec`, то сообщения об ошибке будут менее понятными. Кроме того, нелишне напомнить, что команды `\begin` и `\end`, ограничивающие окружение, ограничивают группу: все неглобальные определения и изменения параметров, происходящие внутри окружения, забываются по выходе из него.

Новые окружения можно определять так, чтобы они принимали аргументы. Пусть, например, в зависимости от обстоятельств нам нужны рамки разной ширины. Тогда разумно модифицировать определение окружения `ramka` таким образом, чтобы ширина текста в рамке передавалась ему как аргумент. Соответствующее определение будет выглядеть так:

```
\newenvironment{ramka}[1]{\begin{tabular}{|p{#1}|}
                          \hline}{\\\hline\end{tabular}}
```

После этого можно писать, например,

```
\begin{ramka}{6cm}
Текст...
\end{ramka}
```

или даже

```
\begin{ramka}{.85\textwidth}
Текст...
\end{ramka}
```

Общие правила таковы. Чтобы создать окружение с аргументами, надо воспользоваться командой `\newenvironment` с необязательным аргументом. Этот необязательный аргумент ставится между первым и вторым обязательными; как и в случае с `\newcommand`, он означает количество аргументов, которые будет требовать окружение, и это количество не может превышать девяти; места, куда будут вставлены аргументы, по-прежнему обозначаются `#1`,…,`#9`, причем эти значки можно употреблять *только в открывающих командах* (т.е. во втором обязательном аргументе команды `\newenvironment`)

С помощью `\newenvironment` нельзя переопределить уже существующее окружение (если вы все же попробуете так сделать, LaTeX выдаст сообщение об ошибке). Если вам действительно необходимо такое переопределение, надо пользоваться командой `\renewenvironment`, работающей точно так же, как и `\newenvironment`, с тем различием, что в качестве первого аргумента ей можно передавать только имя уже существующего окружения.

У команд для (пере)определения окружения также существуют «варианты со звездочкой»: `\newenvironment*` и `\renewenvironment*`. Если окружение с аргументами определено с помощью одной из этих команд, то в его аргументе запрещены пустые строки или команды `\par`.

5.2. Окружения типа «теорема»

Если вы пишете математический текст, то в этом тексте будет содержаться немалое количество теорем, лемм, определений и тому подобных вещей. Эти элементы математического текста желательно оформлять специальным образом. Например, формулировки теорем часто печатают, для ясности, другим шрифтом, само слово «теорема» также выделяют (третьим) шрифтом и т. д. Чтобы задать такое оформление, в исходном тексте приходится написать довольно много TeX'овских команд, и лучше не повторять этот длинный набор команд много раз, а создать заменяющее его макроопределение, что, в свою очередь, может потребовать некоторого труда (чем-то подобным мы занимались в предыдущих разделах, когда разрабатывали команду `\z`). Если же вы или редакция, с которой вы имеете дело, не слишком требовательны к деталям оформления, то соответствующие макросы (точнее говоря, новые окружения) легко создать из полуфабрикатов, предоставляемых нам для этих целей LaTeX'ом.

Окружения, используемые в LaTeX'е для оформления фрагментов текста типа «теорема», заранее не определены. Дело в том, что количество различных типов объектов наподобие теоремы, присутствующих в одном тексте, может быть достаточно велико (предложение, утверждение, лемма, определение, замечание,…), так что LaTeX в целях экономии машинной памяти и исходя из того, что на все вкусы таких окружений все равно не напасешься, определять их предоставляет вам. Как это делать, удобнее всего разобрать на примере.

Пусть в нашем тексте присутствуют «предложения». Давайте создадим окружение `predl` таким образом, чтобы можно было, например, писать

Предложение 1. *Волга впадает в Каспийское море.*

Доказательство. См. любую географическую карту.

```
\begin{predl}
Волга впадает в Каспийское море.
\end{predl}
\textbf{Доказательство.}
См. любую географическую карту.
```

Для создания такого окружения используется команда `\newtheorem`:

```
\newtheorem{predl}{Предложение}
```

Как видите, команда `\newtheorem` имеет два обязательных аргумента: первый — название окружения, которое мы создаем, второй — заголовок нашей «теоремы».

Теперь обсудим, как работают окружения, созданные при помощи команды `\newtheorem` (будем называть их просто окружениями типа «теорема»). Во-первых, как вы уже заметили, формулировка печатается курсивом, а заголовок — полужирным шрифтом. Во-вторых, абзац, идущий после нашего окружения, начинается с абзацным отступом, если после закрывающей окружение команды `\end` идет пустая строка, и без отступа в противном случае (так что в этом отношении окружения типа «теорема» ведут себя совершенно аналогично таким окружениям, как `quote`, `itemize` и т. п.). В-третьих, окружение типа «теорема» может иметь необязательный аргумент (как обычно, в квадратных скобках). Текст, стоящий в этих квадратных скобках, будет напечатан в скобках после заголовка «теоремы» и ее номера. Обычно это используется для указания ученого, чьим именем названа «теорема»:

Предложение 2 (Пифагор). *Пифагоровы штаны на все стороны равны.*

```
\begin{predl}[Пифагор]
Пифагоровы штаны на
все стороны равны.
\end{predl}
```

При пользовании классами документов, предоставляемыми Американским математическим обществом, появляются дополнительные возможности влиять на оформление «теорем». См. следующий пункт.

Вместе с окружением типа «теорема» автоматически создается и счетчик, хранящий его номер. Имя этого счетчика совпадает с именем окружения (так что в нашем примере счетчик называется `predl`); чтобы изменить представление на печати номеров наших «теорем», можно обычным образом переопределить соответствующую the-команду. Например, если мы хотим, чтобы предложения нумеровались прописными латинскими буквами, надо в преамбуле написать:

```
\renewcommand{\thepredl}{\Alph{predl}}
```

«Теоремы», определяемые описанным выше способом, будут иметь сплошную нумерацию на протяжении всего документа. Как мы уже понимаем, это далеко не всегда удобно. Часто хотелось бы сделать так, чтоб, например, в каждом разделе нумерация «теорем» начиналась заново. Для таких целей предусмотрена команда `\newtheorem` с необязательным аргументом. Этот аргумент ставится после двух обязательных и представляет собой имя того счетчика, которому будет подчинен счетчик нашей «теоремы». Пусть, например, в нашем тексте есть не только предложения, но и теоремы (без кавычек), и мы хотим, чтобы нумерация теорем начиналась заново в каждом разделе. Тогда можно написать в преамбуле так:

```
\newtheorem{theorem}{Теорема}[section]
```

После этого можно будет писать, например, вот что:

Теорема 5.1. *Сумма углов треугольника равна* 180°.

```
\begin{theorem}
Сумма углов треугольника
равна $180^{\circ}$.
\end{theorem}
```

Обратите внимание, что, если «теорема» определена таким образом (со счетчиком, подчиненным другому счетчику), то представление ее номера на печати изменяется: при определении

```
\newtheorem{xyz}[abcd]
```

(счетчик «теоремы» типа `xyz` подчинен счетчику `abcd`) команда `\thexyz` будет определена как

```
\theabcd.\arabic{xyz}
```

(если вы хотите, чтобы нумерация «теоремы» представлялась на печати иначе, вы опять-таки можете переопределить the-команду).

Наконец, LaTeX предоставляет еще одну возможность нумерации определяемых вами «теорем». Предположим, что кроме теорем в вашем тексте есть еще и леммы, и при этом вы хотите, чтобы леммы и теоремы нумеровались совместно: теорема 2.1, теорема 2.2, затем лемма 2.3, затем теорема 2.4 и т. д. Тогда, предполагая, что окружение `theorem` уже определено, как выше, можно определить окружение `lemma` так:

```
\newtheorem{lemma}[theorem]{Лемма}
```

В этом случае необязательный аргумент команды `\newtheorem` располагается между двумя обязательными; этот аргумент — имя того окружения типа «теорема», совместно с которым будет нумероваться определяемая вами «теорема».

Команду `\newtheorem` можно использовать или с одним необязательным аргументом, или с другим, но не с обоими вместе.

5.3. Окружения типа «теорема» в пакете amsthm

Все LaTeX'овские «теоремы», определяемые пользователем при помощи окружения `newtheorem`, оформляются в одном и том же стиле, что не всегда приемлемо. Пакет `amsthm`, распространяемый Американским математическим обществом, позволяет внести в это оформление некоторое разнообразие. Итак, предположим, что вы его подключили. Что нового, по сравнению с «чистым» LaTeX'ом, можно сделать?

Во-первых, в этом пакете определен «вариант со звездочкой» команды `\newtheorem`. Именно, если определить очередной тип «теорем» с помощью `\newtheorem*` вместо `\newtheorem`, то «теоремы» указанного типа не будут нумероваться.

Во-вторых, для управления стилем оформления окружений типа «теорема» предназначена команда `\theoremstyle` , аргументом (единственным) которой может быть слово `plain`, `definition` или `remark`. Если в преамбуле дать эту команду с одним из трех допустимых аргументов, то все «теоремы», определяемые с помощью `\newtheorem` после этой команды `\theoremstyle`, будут оформлены в соответствующем стиле; чтоб определить тип теорем, оформляемый в другом стиле, надо написать еще одну команду `\theoremstyle` (с другим аргументом, разумеется), а уж после нее — очередную `\newtheorem`. Стиль `plain` рекомендуется для собственно теорем, предложениий и лемм, `definition` — для определений, `remark` — для замечаний[4]. Если в преамбуле нет ни одной команды `\theoremstyle`, подразумевается стиль `plain`.

В-третьих, в пакете `amsthm` предусмотрено также окружение `proof`, предназначенное для оформления доказательств. Это окружение автоматически ставит слово **Proof** в начало доказательства и автоматически же завершает доказательство символом □. Если вас не устраивает, что слово «доказательство» пишется по-английски, нужно переопределить с помощью `\renewcommand` команду `\proofname` (ср. с. 169). Ес-

[4] В стиле `plain` заголовок печатается жирным шрифтом, а текст «теоремы» — курсивом, в стиле `definition` заголовок печатается жирным шрифтом, а текст «определения» — прямым, в стиле `remark` заголовок печатается курсивом, а текст «замечания» — прямым шрифтом.

ли символ □ нужен вам сам по себе (например, как знак завершения какого-то рассуждения, не выделенного в качестве доказательства нумерованного утверждения), можно воспользоваться командой `\qed`.

Окружение `proof` допускает и необязательный аргумент: если написать, скажем,

```
\begin{Proof}[Доказательство основной теоремы]
```

то вместо слова **Proof** появится текст, записанный нами в квадратных скобках.

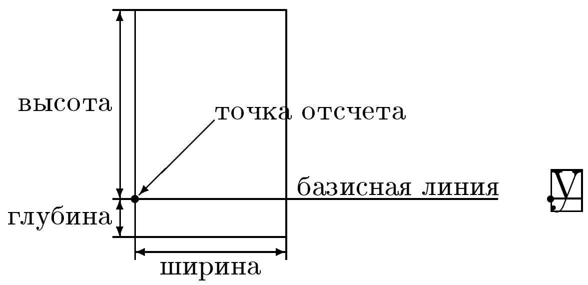
Глава VIII

Блоки и клей

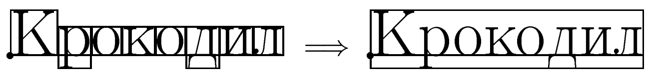
1. Текст состоит из блоков

Мы уже отмечали, что в процессе набора TeX не принимает во внимание, как буквы будут выглядеть на печати, а лишь учитывает, сколько места надо отвести на каждый символ. Давайте обсудим этот процесс подробнее.

С точки зрения TeX'а, каждая буква представляет собой *блок* (английский термин: box), т. е. прямоугольник с выделенной *точкой отсчета*; горизонтальная прямая, проходящая через точку отсчета, называется *базисной линией* (английский термин: baseline). Блок характеризуется тремя размерами: шириной, высотой и глубиной. См. рисунок, на котором также изображен блок, соответствующий букве у.



Когда из букв составляются слова (а из слов — строки), блоки, соответствующие отдельным буквам, ставятся рядом так, чтобы их базисные линии были продолжением друг друга. Каждая строка также становится блоком, точка отсчета которого совпадает с точкой отсчета первого из составляющих ее блоков:



Страницы — это тоже блоки, составленные из блоков, соответствующих строкам. Эти блоки ставятся таким образом, чтобы точки отсчета были одна над другой, после чего в качестве точки отсчета и базисной линии полученного блока берутся точка отсчета и базисная линия последнего из добавляемых блоков:

Страница, составленная из строк, становится блоком, в котором точки отсчета. . .	$\Longrightarrow$	Страница, составленная из строк, становится блоком, в котором точки отсчета. . .

В приведенных выше примерах мы сталкивались с блоками, которые TeX создает автоматически; в настоящей главе пойдет речь о командах, предназначенных для создания блоков вручную. Сначала мы расскажем, какие средства для этого предоставляет нам LaTeX, а затем рассмотрим некоторые TeX'овские команды, дающие дополнительные возможности.

2. Команды LaTeX'а для генерации блоков

2.1. Блоки из строки́

С одной командой для генерации блоков мы уже знакомы: это команда `\mbox`. Эта команда создает блок из текста, набираемого в одну строку. Полученный блок рассматривается TeX'ом как одна буква:

Проказница мартышка,
осел, козел и косолапый
мишка. . .

```
Проказница мартышка,
\mbox{осел, козел} и
косолапый мишка\ldots
```

В этом примере TeX никогда не разорвет строку между словами «осел» и «козел» и никогда не сделает переносов в этих словах: при верстке абзаца TeX имеет дело не с этими словами по отдельности, а только с блоком, в который входят они оба вместе с пробелом между ними. По той же причине TeX не сможет растянуть или сжать пробел между словами «осел» и «козел» для выравнивания строк в абзаце.

Теперь, когда мы знаем, что такое TeX'овские блоки, можно признаться, что окружения `picture`, `tabular` и `array` тоже генерируют блоки, и именно поэтому создаваемый ими текст воспринимается TeX'ом как одна большая буква.

В аргументе команды `\mbox` может присутствовать все то же, что может быть в обычном тексте в пределах одной строки: математические формулы, команды смены шрифта или присваивания значений каким-то параметрам, команды для генерации блоков (например, тот же `\mbox`, или даже окружения `picture` или `array`) и т. д. Запрещены в аргументе команды `\mbox` пустые строки или команды `\par`, выключные математические формулы, окружения, определяющие абзацы специального вида (скажем, `itemize` или `center`), команда `\\` и тому подобные вещи, «не вписывающиеся в строку». Если в аргументе команды `\mbox` происходит смена шрифта, изменение каких-то параметров или определение команд, то по выходе из блока все эти изменения забываются, поскольку фигурные скобки, ограничивающие аргумент команды `\mbox`, ограничивают также и группу («глобальные» команды вроде `\setcounter` сохраняют свое действие и по выходе из блока).

Блок, создаваемый командой `\mbox`, имеет ширину, равную «естественной» длине строки текста, являющегося его аргументом. Можно также создать блок из строки текста, ширина которого отлична от ее естественной длины. Для этого используется команда `\makebox`. Эта команда имеет один обязательный аргумент, имеющий такой же смысл, как аргумент команды `\mbox`, и, кроме того, необязательный аргумент — ширину блока, порождаемого командой:

Туда и обратно. `Туда \makebox[5em]{и} обратно.`

Как видите, необязательный аргумент ставится перед обязательным; длина в нем может быть указана, как обычно, либо в какой-либо из TeX'овских единиц, либо как параметр со значением длины, возможно — с числовым коэффициентом (см. разд. VII.4). Сам текст, являющийся обязательным аргументом команды `\makebox`, размещается по центру в блоке ширины, указанной в необязательном аргументе. Если указать в необязательном аргументе команды `\makebox` ширину, меньшую естественной длины строки, то текст выйдет за края блока; поскольку место, отводимое TeX'ом блоку, определяется только тем, каковы ширина, высота и глубина блока, а не тем, какие размеры реально имеет текст, содержащийся в блоке, при этом может возникать наложение одного текста на другой. Например, размеры и точка отсчета блока, создаваемого командой `\makebox[1.5em]{123456}`, выглядят с точки зрения TeX'а так:

123456

Для ясности мы использовали в этом примере крупный шрифт. А вот как такой «выпирающий за края» блок взаимодействует с окружающим текстом:

текст123456текст `текст\makebox[1.5em]{123456}текст`

Можно также создать блок заданной ширины, в котором текст будет не центрирован, а прижат к правому или левому краю (полиграфисты говорят: «выключен вправо или влево»). Для этого в команде `\makebox` предусмотрен второй необязательный аргумент — буква `l` для текста, выключенного влево, или `r` для текста, выключенного вправо (можно также указать аргумент `c` — тогда текст будет центрирован, так же, как если бы второго необязательного аргумента не было). Пример:

```
        текст
         екст
          кст
  текст
текст
```

```
\parindent=0pt
\makebox[10em][r]{текст}\\
\makebox[10em][r]{екст}\\
\makebox[10em][r]{кст}\\
\makebox[10em][c]{текст}\\
\makebox[10em][l]{текст}
```

Мы установили нулевое значение абзацного отступа, чтобы все строки, включая первую, начинались с самого начала. Кстати, обратите внимание, что у нас получилась верстка с выравниванием без помощи таких вещей, как `tabbing` или `tabular`.

У команды `\makebox` значение ширины блока можно установить равным нулю. Если при этом присутствует необязательный аргумент `l`, то получится блок нулевой ширины, а текст будет выходить за его пределы вправо (и, стало быть, наложится на последующий текст в строке, если таковой присутствует); если присутствует необязательный аргумент `r`, то текст будет выходить влево за пределы блока (и тем самым накладываться на предшествующий текст):

текст???кст	`текст\makebox[0pt][l]{???}текст\\`
тек???текст	`текст\makebox[0pt][r]{???}текст\\`

Наряду с `r` («прижатый вправо»), `l` («прижатый влево») и `c` («центрированный»), в качестве второго необязательного аргумента команды `\makebox` можно использовать и букву `s`, с которой начинаются английские слова stretched (растянутый) и shrunk (ужатый). Соответственно, при указании такого второго необязательного аргумента текст будет равномерно растянут или сжат до ширины, указанной в первом необязательном аргументе. Если при этом придется превысить предел растяжимости, то появится сообщение `Underfull \hbox`, а если окажется, что превышен предел сжимаемости, то вы увидите сообщение `Overfull \hbox`. Чтобы осмысленно применять `\makebox` с необязательным аргументом `s`, надо уметь управлять растяжимостью и сжимаемостью промежутков. Как это делать, рассказано в следующем разделе.

До сих пор мы задавали ширину блока в команде `\mbox` в явном виде. Можно, кроме того, выразить эту ширину через «естественную» ширину текста. Для этого служит команда `\width`. Вот, например, как сделать, чтобы чтобы ширина блока, получаемого с помощью `\makebox`, была на 30% больше естественной:

скоросшиватель	`\makebox{скоросшиватель}\\[2pt]`
скоросшиватель	`\makebox[1.3\width][r]{скоросшиватель}`

Командой `\width` можно пользоваться только внутри необязательного аргумента `\makebox` (или `\framebox` — см. ниже). Не пытайтесь пользоваться ею в качестве параметра со значением длины — ничего хорошего из этого не выйдет.

2.2. Блоки из абзацев

Если необходимо создать блок, в котором размещается сверстанный TeX'ом абзац текста, то можно воспользоваться командой `\parbox`. У этой команды два обязательных аргумента: первый — длина строк в получаемом абзаце, второй — собственно текст. Например, такой текст

В строку вставили целый абзац текста, сверстанного по всем TeX'овским правилам. После этого продолжается прерванная строка.

получился следующим образом:

```
В строку\qquad
\parbox{4cm}{вставили целый
абзац текста, сверстанного
по всем \TeX'овским правилам.
После этого продолжается}\qquad
прерванная строка.
```

Как видите, базисная линия блока, создаваемого командой `\parbox`, находится в точности посредине текста. Поэтому команду `\parbox` удобно использовать для включения больших фрагментов текста в математические формулы. Например, формула

$$\int_a^b f'(x)\,dx=f(b)-f(a)\qquad \parbox{4cm}{для всех функций f, производная которых интегрируема по Риману}$$

получается из такого исходного текста:

```
$$
  \int_a^b f'(x)\,dx=f(b)-f(a)\qquad
  \parbox{4cm}{для всех функций $f$,
  производная которых интегрируема
  по Риману}
$$
```

Если дать команду `\parbox` с необязательным аргументом, то создаваемый ею блок можно расположить относительно строки и по-иному: чтобы вровень с остальной строкой шла самая верхняя строка абзаца (для этого нужен аргумент `t`) или самая нижняя (аргумент `b`); можно также указать аргумент `c` — тогда блок будет расположен по центру, так же, как если бы необязательного аргумента вообще не было. Необязательный аргумент у этой команды должен идти перед обязательными.

Во втором обязательном аргументе команды `\parbox`, задающем текст, может присутствовать всё то же, что в обычном тексте, в том числе команды для вертикальных пробелов наподобие `\vspace`, пустые строки, разделяющие абзацы, выключные формулы и т. п. Абзацы, создаваемые командой `\parbox`, по умолчанию делаются без абзацного отступа и в режиме `\sloppy`. Если вы хотите чего-то другого, можно прямо внутри аргумента команды `\parbox` установить нужное вам значение абзацного отступа, параметра `\tolerance` и т. п. (см. разд. III.6 по поводу смысла этих параметров).

Можно также указать LaTeX'у высоту, которую должен иметь блок, полученный в результате применения команды `\parbox`. Для этого используется второй необязательный аргумент, идущий непосредственно после первого. Наряду с явным указанием размера, можно воспользоваться командой `\height`, обозначающей «естественную» высоту текста, а также командой `\totalheight` (высота плюс глубина).

Наконец, в команде `\parbox` можно указать, как именно должен быть расположен текст внутри блока. Для этого используется третий необязательный аргумент, следующий непосредственно после второго. Этот аргумент — буква `t`, `b`, `c` или `s`. Буква `t` означает «сверху», `b` — «снизу», `c` — «по центру». Если третий необязательный аргумент не указан, то по умолчанию считается, что он совпадает с первым. Если же третьим необязательным аргументом является `s`, это означает, что текст будет растянут или ужат в соответствии с размером, указанным во втором необязательном аргументе. Если вы не позаботитесь о специальных командах, обеспечивающих такую растяжимость или сжимаемость, то получите сообщение об Overfull'е или Underfull'е.

В следующем примере блоки, созданные командой `\parbox`, для наглядности взяты в рамку (с помощью `\fbox`):

Мы едем, едем, едем в далекие края.

Мы едем, едем, едем в далекие края.

```
\fbox{%
\parbox[b][1.3\height]{2cm}{%
Мы едем, едем, едем
в далекие края.}}\qquad
\fbox{%
\parbox[t][1.3\height]{2cm}{%
Мы едем, едем, едем
в далекие края.}}
```

Команды `\height` и `\totalheight`, так же как и `\depth`, можно использовать только в необязательном аргументе команды `\parbox` (а также `\framebox` или `\makebox`).

Наряду с `\parbox`, существует еще один способ создать блок из абзацев. Именно, существует окружение `minipage` («министраница»), генерирующее блок из текста, расположенного внутри этого окружения; блок состоит из абзацев, ширина которых задается в обязательном аргументе окружения `minipage` (так же, как в команде `\parbox`); перед обязательным аргументом этого окружения может стоять необязательный: буква `t`, `b` или `c`, причем смысл этого аргумента опять-таки такой же, как в команде `\parbox`. Основное отличие `minipage` от `\parbox` в том, что к тексту внутри этого окружения можно делать сноски с помощью команды `\footnote`, причем текст сноски появляется не внизу страницы, а внизу блока, генерируемого окружением `minipage`. При наборе книги, которую вы читаете, это окружение использовалось для печати примеров.

2.3. Текст в рамке; комбинации блоков

В гл. III мы уже упоминали про команду `\fbox`, берущую в рамку фрагмент текста, помещающегося в строку. Наряду с ней есть и команда `\framebox`, относящаяся к ней так же, как `\makebox` относится к `\mbox`: она берет текст в рамку заданного размера, причем текст внутри этой рамки либо центрирует (если необязательного аргумента нет или же задан необязательный аргумент `c`), либо прижимает к правому или левому краю рамки (если задан необязательный аргумент `r` или `l`). Смысл и расположение обязательных и необязательных аргументов у команды `\framebox` такой же, как и у команды `\makebox`.

Точнее говоря, первый обязательный аргумент команды `\framebox` задает не ширину рамки, а ширину текста, помещаемого в эту рамку. Сама же рамка отделена от текста пробелом ширины `\fboxsep`; толщина линий в рамке равна `\fboxrule`. Обоим этим параметрам можно обычным образом присваивать новые значения (см. разд. VII.4).

Коль скоро каждый блок, создаваемый LaTeX'овскими командами, рассматривается TeX'ом просто как большая буква, возможны любые, сколь угодно причудливые, комбинации таких «букв». Пусть, например, нам надо взять в рамку абзац текста шириной 6 см, чтобы получилось так:

Внутри TEX'овских блоков может присутствовать не только собственно текст или формулы, но и другие блоки, внутри этих блоков — еще блоки, и так далее. Таким образом, блоки могут быть вложены друг в друга, как матрешки.

Просто поместить этот текст в аргумент команды `\fbox` не получится, поскольку наш текст в одну строку не укладывается, а команда `\fbox`, подобно команде `\mbox`, текстов, не укладывающихся в строку, не переваривает. Поэтому нужно сделать из нашего абзаца блок с помощью команды `\parbox` и этот блок (т. е. уже «букву») передать в качестве аргумента команде `\fbox`:

```
\fbox{%
\parbox{6cm}{%
Внутри \TeX'овских блоков может ...
... друг в друга, как матрешки.}%
}
```

Обратите внимание на знаки процента, которыми заканчиваются первая и предпоследняя строки. Если бы их не было, то рамка отстояла бы от текста больше, чем надо, так как TEX решил бы, что аргумент команды `\fbox` имеет пробел до и после «буквы», созданной командой `\parbox`. См. с. 18 по поводу использования знака процента для удаления нежелательных пробелов.

2.4. Сдвиги относительно базисной линии

Когда при исполнении команды `\makebox` или `\mbox` TEX создает блок из меньших блоков (каждая буква, как мы помним, — это блок, из букв составляются слова — тоже блоки; наконец, блоки могут быть заданы в явном виде, в частности, командой `\mbox`), то блоки эти размещаются в строке таким образом, что все их точки отсчета расположены на одной высоте (иными словами, их базисные линии продолжают одна другую). Можно, однако, сдвинуть блок по вертикали относительно базисной линии. Для этого удобно воспользоваться LATEX'овской командой `\raisebox`. Эта команда требует двух обязательных аргументов.

Первый из них — расстояние, на которое сдвигается по вертикали фрагмент текста, второй — сам этот фрагмент текста. Пример:

Слово $^{\text{подскочило}}$ в строке.

```
Слово \raisebox{2pt}{подскочило}
в строке.
```

Текст, расположенный во втором обязательном аргументе этой команды, должен удовлетворять тем же требованиям, что и аргумент команды `\mbox`: в нем могут быть самые разные TeX'овские команды, при условии, что среди них не будет команд типа пустой строки, `\par`, `\\` и тому подобных, которые «не укладываются в строку» (зато в этом тексте, как водится, могут присутствовать любые команды, порождающие блоки, в частности, например, `\parbox`, а уж в ее аргументе оставляйте пустых строк, сколько душе угодно). Если первый обязательный аргумент команды `\raisebox` отрицателен, то текст будет, естественно, не поднят, а опущен. Вот, например, как можно определить команду `\TeX`, печатающую эмблему TeX'а:

```
\newcommand{\TeX}{T\nolinebreak\hspace{-.1667em}\raisebox
          {-.5ex}{E}\nolinebreak\hspace{-.125em}X}
```

Тут же мы видим и примеры использования отрицательных промежутков для того, чтобы буквы сблизились. Команды `\nolinebreak` нужны, чтобы не случилось разрыва строки посередине эмблемы.

На самом деле команда `\TeX` определяется более экономным способом, который требует меньше машинного времени и памяти, но использует не рассматриваемые нами средства TeX'а. Время от времени мы будем приводить определения команд «в переводе с TeX'а на LaTeX».

Кроме вертикального сдвига блоков, команда `\raisebox` может делать еще одно полезное дело: с ее помощью можно обмануть TeX, заставив его считать, что блок, полученный после сдвига, имеет любую заданную нами высоту и глубину, независимо от того, сколько места реально занимает текст. Именно, эта команда может принимать, наряду с обязательными, необязательные аргументы. Между двумя обязательными аргументами можно указать необязательный аргумент — высоту, которую, по мнению TeX'а, должен иметь сдвинутый блок. Кроме того, после первого необязательного аргумента может стоять второй — глубина, которую, по мнению TeX'а, будет иметь сдвинутый блок. Вот пример:

Строка.
Вторая Ы

Третья строка.

```
Строка.\\
Вторая
\raisebox{7pt}[1pt][10pt]{Ы}\\
Третья строка.
```

Буква Ы, поднятая на 7 пунктов над строкой, наложилась на первую строку, поскольку в первом необязательном аргументе команды `\raisebox` мы приказали TEX'у считать, что блок, образованный поднятой буквой Ы, имеет высоту всего лишь один пункт (стало быть, возвышается над базисной линией второй строки меньше, чем любая буква), и соответственно TEX не сделал дополнительного интервала между первой и второй строками. С другой стороны, третья строка отодвинулась от второй, поскольку во втором необязательном аргументе команды `\raisebox` мы велели TEX'у считать, что глубина блока, образованного поднятой буквой Ы, равна целым десяти пунктам, и TEX послушно оставил дополнительное место, чтобы этот блок не наложился на третью строку!

Иногда разумно использовать команду `\raisebox` даже с нулевым обязательным аргументом, только для того, чтобы менять (в глазах TEX'а) высоту и/или глубину блока, не сдвигая его относительно базисной линии. В гл. IX мы увидим пример такого использования этой команды.

3. Команда `\hbox`

Возможности, предоставляемые LATEX'ом для генерации блоков, достаточны для простых приложений, но в более серьезных случаях их не хватает. В этом и следующем разделах мы рассмотрим более гибкие средства, предоставляемые для этой цели непосредственно языком TEX и макропакетом Plain TEX. Мы не будем пытаться описать все TEX'овские команды для генерации блоков (книгу [2] ничто заменить не может), но сообщим тот минимум сведений, который необходим для модификации LATEX'овского стандартного оформления, о чем пойдет речь в следующей главе. Подчеркнем, что всеми описываемыми в этом и следующем разделах TEX'овскими средствами можно пользоваться в LATEX'овских исходных текстах.

Прежде всего давайте вспомним (с. 141), что в каждый момент трансляции исходного текста TEX находится в одном из трех следующих режимов: горизонтальном (в процессе верстки абзаца), вертикальном (между абзацами), или математическом (в процессе набора математической формулы); при появлении первой же буквы или LATEX'овской команды для генерации блока или линейки (к таковым относятся `\mbox`, `\makebox`, `\fbox`, `\framebox`, окружения `array`, `tabular` или `picture`, а также команда `\rule`[1]) TEX из вертикального режима выходит и начинает очередной абзац.

[1]Но не `\hrule` или `\vrule`: это команды TEX'а, а не LATEX'а.

Одна из основных TeX’овских команд для генерации блоков называется `\hbox`. В своем простейшем виде она полностью аналогична LaTeX’овской команде `\mbox`, с одним важным отличием: в вертикальном режиме команда `\hbox` *не* начинает нового абзаца, а только добавляет сгенерированный ею блок (т. е. фактически строку) к уже сверстанной части страницы. Внутри абзаца (в горизонтальном режиме) команда `\hbox` действует точно так же, как и `\mbox`. Вот пример:

На странице уже присутствует абзац текста. После того, как он кончится, TeX перейдет в вертикальный режим.
Строка
Еще строка
Только теперь начинается новый абзац.

```
На странице \hbox{уже}
присутствует абзац текста.
После того, как он
кончится, \TeX{} перейдет в
вертикальный режим.

\hbox{Строка} \hbox{Еще строка}
Только
теперь начинается новый абзац.
```

Сравните с тем, что было бы при использовании LaTeX’овской команды `\mbox` вместо `\hbox`:

На странице уже присутствует абзац текста. После того, как он кончится, TeX перейдет в вертикальный режим.
Эти слова сразу начинают новый абзац.

```
На странице уже присутствует
абзац текста. После того, как
он кончится, \TeX{} перейдет
в вертикальный режим.

\mbox{Эти слова} сразу
начинают новый абзац.
```

3.1. Растяжимые интервалы

До сих пор шла речь о важных, но непринципиальных различиях между TeX’овским `\hbox` и LaTeX’овским `\mbox`. Теперь поговорим о дополнительных возможностях, предоставляемых TeX’овской командой.

Команда `\hbox` «в чистом виде» создает блок, ширина которого равна естественной длине текста, являющегося ее аргументом. Кроме этого, она может создавать блоки любой заданной ширины. Для этого нужно сказать

`\hbox to` ⟨*ширина*⟩ `{`*текст*`}`

Здесь ⟨*ширина*⟩ должна быть выражена в воспринимаемых TeX’ом единицах длины: это может быть, например, `20pt`, или `2.3cm`, или, например, `0.12\textwidth` — параметр со значением длины (возможно, с коэффициентом) тоже годится. Между `to` и обозначением ширины, а также между обозначением ширины и открывающей фигурной скобкой могут быть пробелы — TeX их проигнорирует[2]. Наконец, отсутствие backslash в слове `to` не является опечаткой: это не команда, а одно из «ключевых слов» TeX'а (подобно ключевым словам `plus` и `minus`, с которыми мы вскоре снова встретимся, или `width` и `height`, с которыми мы уже встречались в разделе, посвященном линейкам). Давайте опробуем эту новую возможность команды `\hbox`:

Два слова	`\hbox to 3cm {Два слова}`

Если вы опробовали этот пример на вашем компьютере, то заметили, что на экране появилось сообщение

```
Underfull \hbox
```

Дело в том, что пробел между словами «Два» и «слова» не может растянуться настолько, чтобы наш блок имел ширину три сантиметра; в ситуациях, когда пробел насильно заставляют растянуться больше, чем положено, возникает сообщение об `Underfull`'е, как это было объяснено в разд. III.6.6.

Можно, однако, заставить TeX создать блок требуемой ширины «без скандала». Для этого в том промежутке, который мы хотим растянуть, надо поставить команду `\hfil`:

Два слова	`\hbox {Два слова}`
Два слова	`\hbox {Два \hfil слова}`
Два слова	`\hbox to 2cm {Два \hfil слова}`
Два слова	`\hbox to 3cm {Два \hfil слова}`
Два слова	`\hbox to 4cm {Два \hfil слова}`

Если мы не указываем явно ширину блока, а предоставляем TeX'у создать блок «естественной» ширины, то команда `\hfil` никакого действия не оказывает; если промежуток для достижения требуемой ширины надо растянуть, то растяжение на требуемое расстояние будет проведено в том месте, где стоит команда `\hfil`.

Если в аргументе команды `\hbox` присутствует не одна команда `\hfil`, а несколько, то растяжение произойдет на месте каждой из этих команд, причем размер этого растяжения будет распределен между командами `\hfil` равномерно: если необходимо превысить естественную ширину блока на 5 см, а в аргументе команды `\hbox` стоят два `\hfil`, то на месте каждого из них будет добавлен пробел в 2,5 см. Вот пример с несколькими `\hfil`:

[2]Пустых строк, однако, быть не должно.

Раз два три `\hbox to 4cm{Раз \hfil два \hfil три}`

В частности, если `\hfil` стоит справа или слева от текста, то весь текст будет прижат влево или вправо, поскольку `\hfil` отмечает то единственное место, в котором интервалы могут растягиваться; если же две команды `\hfil` стоят по обе стороны от текста, то текст внутри блока будет центрирован, поскольку дополнительное растяжение поделится между двумя `\hfil` поровну:

Слева	`\hbox to 0.7\textwidth {Слева\hfil}`
Справа	`\hbox to 0.7\textwidth {\hfil Справа}`
В центре	`\hbox to 0.7\textwidth {\hfil В центре\hfil}`

Можно считать, что на месте каждого `\hfil` в строку вставляется пружина; все эти пружины имеют одинаковую жесткость, в свободном состоянии все они имеют нулевую ширину, и все эти пружины могут сколь угодно широко растягиваться.

Наряду с `\hfil` существует команда `\hfill`, также задающая бесконечно растяжимые пробелы, причем эта растяжимость «в бесконечное число раз больше», чем у пробелов, задаваемых `\hfil`. Если в аргументе команды `\hbox` присутствуют `\hfil` и `\hfill` совместно, то все растяжения происходят только за счет «более растяжимых» `\hfill`:

Слово	`\hbox to 4cm{\hfil Слово\hfil}`
Слово	`\hbox to 4cm{\hfill Слово\hfil}`
Слово	`\hbox to 4cm{\hfil Слово\hfill}`

3.2. Лидеры

В оглавлении к этой книге (и ко многим другим тоже) место между названием раздела и номером страницы заполняется рядом из точек. Это можно сделать с помощью LaTeX'овской команда `\dotfill`. Она работает так же, как и `\hfill`, с той разницей, что пробел, образующийся в результате действия этой команды, заполняется точками:

А . . . . Б `\hbox to 3cm{А\dotfill Б}`

Кроме этого, есть LaTeX'овская команда `\hrulefill`, которая также действует аналогично команде `\hfill` и при этом заполняет пробел линейкой:

1__________2__________3

```
\hbox to 5cm{1\hrulefill
2\hrulefill 3}
```

В TEXнической терминологии такие заполнители называют лидерами (leaders).

На самом деле можно заполнить пробел не только точками или линейкой, но и любым повторяющимся текстом. Вот как это делается. Пусть мы хотим заполнить пробел повторяющимися твердыми знаками. Тогда можно написать так:

1 ЪЪЪЪЪЪЪЪЪЪЪ 2

```
\hbox to 5cm{1\leaders
\hbox{Ъ}\hfil 2}
```

Если бы мы хотели, чтоб буквы Ъ шли не вплотную, можно было бы, например, вместо `\hbox{Ъ}` написать так:

```
\hbox to 2em{\hfil Ъ\hfil}
```

В общем случае применяйте команду `\leaders` так:

`\leaders` ⟨*блок*⟩ ⟨`\hfil` *или* `\hfill`⟩

Здесь ⟨*блок*⟩ — это любая TEX'овская команда для генерации блока, например, `\hbox`, с которой мы уже познакомились, или `\vbox` или `\copy`, о которых еще пойдет речь. Команды LATEX'а (`\mbox`, `\makebox`, `\parbox` и т. п.) применять в этом контексте нельзя; если, тем не менее, хочется воспользоваться их возможностями, то их надо «спрятать» в `\hbox`, написав, например,

```
\hbox{\makebox[3em][r]{...}}
```

Между командой для генерации блока и командой `\hfil` или `\hfill` может быть пробел (например, конец строки). Команда `\leaders` работает так: выделяется столько свободного места, сколько получилось бы, если бы стояло просто `\hfil` или `\hfill`, а затем это место заполняется идущими вплотную друг к другу копиями ⟨*блок*⟩ столько раз, сколько этот блок поместится по ширине на выделенное место (если ширина свободного места меньше ширины блока, то ни разу).

С помощью команды `\leaders` можно также изменить толщину линейки, заполняющей свободное место. Именно, команда `\hrulefill` является по существу сокращением от

```
\leaders\hrule\hfill
```

Если же мы скажем, например,

```
\leaders\hrule height 1pt \hfill
```

то линейка будет иметь толщину 1 пункт, вместо принятых по умолчанию 0.4 пункта. Можно также написать `\hfil` вместо `\hfill`, с очевидными последствиями.

3.3. Клей

Выше мы рассмотрели команды `\hfil` и `\hfill`, которые действуют подобно вставленным в строку пружинам. Можно вставлять в строку пружины с самыми разнообразными свойствами, указав LaTeX'овской команде `\hspace` аргумент, содержащий `plus`- или `minus`-компоненту (в разд. III.9.4 мы упоминали об этой возможности, но в тот момент у нас еще не было серьезных примеров). Именно, если вы скажете

`\hspace{`x `plus` y `minus` z`}`

где x, y и z — длины, то вставите в текст пружину, которая в свободном состоянии имеет длину x, может увеличивать свою длину на y и уменьшать свою длину на z (в отличие от пружин, встречающихся в жизни, может выполняться неравенство $x < z$, и, того пуще, длины y и z могут быть отрицательными, но мы не будем объяснять, как TeX поведет себя в столь странной ситуации).[3] Здесь `plus` и `minus` — это, как мы помним, очередные ключевые слова TeX'а, наподобие `to`, `width` и `height`. Если мы создаем блок естественной ширины, то команда `\hspace` с таким аргументом создаст пробел размером x; если же мы в команде `\hbox` попросим TeX создать блок, ширина которого отличается от естественной, то для достижения требуемой ширины размеры пробелов будут изменяться. В TeXнической терминологии эти «пружины» называются *клеем* (Дональд Кнут отмечает, что название «клей» неудачно, но менять его поздно, поскольку оно, по его словам, «уже прилипло»). Длины y и z, указанные после ключевых слов `plus` и `minus`, называются `plus`- и `minus`-компонентами клея. Длина x называется естественным размером клея. С этой точки зрения команда `\hfil` также помещает в строку клей — с бесконечной растяжимостью и нулевым естественным размером.

Опишем более точно, как именно растягивается или сжимается клей при выполнении команды `\hbox to ...` Для простоты предположим дополнительно, что `plus`- и `minus`-компоненты клея всюду неотрицательны и что в строке отсутствует клей с бесконечной растяжимостью или сжимаемостью (в частности, в строке нет `\hfil`'ов или `\hfill`'ов; про клей с бесконечной сжимаемостью речь пойдет ниже). В этом случае TeX вычисляет «естественную ширину» блока, складывающуюся из ширин составляющих его элементов и естественных размеров клея, и сравнивает ее с требуемой шириной блока, указанной в команде `\hbox` после ключевого слова `to`. Если эти две ширины совпали, то все пробелы будут иметь естественный размер. Если требуемая ширина больше

[3] Если мы заставим такую пружину растянуться или сжаться больше, чем сказано, то получим сообщение «`Underfull \hbox`» или «`Overfull \hbox`»; см. ниже.

естественной, то TeX вычисляет, насколько больше, после чего «разверстывает» эту добавку между всеми пробелами пропорционально величинам `plus`-компонент клея в этих пробелах.

Вот пример. Предположим, мы создаем блок с помощью команды

```
\hbox to a {A\hspace{0pt plus 2em}%
Б\hspace{1cm plus 1em minus 2mm}B}
```

где величина a на 13 мм больше суммы ширин букв А, Б и В, то пробел между А и Б будет равен 2 мм, а пробел между Б и В — 11 мм, поскольку `plus`-компонента клея между А и Б в два раза больше, чем `plus`-компонента клея между Б и В (и никакого другого клея в строке нет, так что ничего более растянуть нельзя). Если требуемая ширина меньше естественной, то уменьшение длины также распределяется между всеми элементами клея пропорционально величинам их `minus`-компонент. Если продолжить аналогию между TeX'овским клеем и пружинами, то можно сказать, что жесткость пружины при растяжении обратно пропорциональна величине `plus`-компоненты.

В приведенном примере оба пробела в блоке были созданы вручную командой `\hspace`; если же в аргументе команды `\hbox` присутствуют пробелы, то следует учесть, что эти пробелы также, как мы объясняли на с. 125, обладают растяжимостью и сжимаемостью, которая также берется в расчет.

В случае, когда пробелы надо растягивать и требуемое растяжение блока больше, чем сумма `plus`-компонент всех элементов клея, на экран и в `log`-файл выдается знакомое вам сообщение `Underfull \hbox`; если пробелы надо уменьшать и величина, на которую надо уменьшить ширину блока, меньше, чем сумма `minus`-компонент всех элементов клея, то выдается не менее знакомое сообщение `Overfull \hbox`.

Все сказанное относилось к случаю, когда бесконечно растяжимого клея в аргументе команды `\hbox` нет. Если же таковой присутствует (например, есть команда `\hfil`) и пробелы надо растягивать, то растяжимость клея с конечными значениями `plus`-компонент утрачивается: соответствующие интервалы будут иметь естественный размер (что бы ни было написано в аргументе команды `\hspace` после `plus`), а все растяжения будут происходить только за счет команд `\hfil`. При этом сообщение об Underfull'е выдаваться не будет, как бы ни растянулись пробелы. Аналогично, если пробелы надо ужимать и присутствует клей с бесконечной сжимаемостью, все уменьшения пробелов произойдут только за его счет и никогда не будет выдано сообщения об Overfull'е.

Если в аргументе команды `\hbox` присутствуют команды `\hfil` и `\hfill` совместно, то при необходимости растягивать пробелы учитывается только `\hfill`, в то время как «в бесконечное число раз менее растяжимый» `\hfil` в расчет не берется (не говоря уж о клее с конечной растяжимостью):

Блоки и клей	`\hbox to 4cm{\hfil Блоки и клей\hfil}`
Блоки и клей	`\hbox to 4cm{\hfil Блоки и клей\hfill}`
Блоки и клей	`\hbox to 4cm{\hfill Блоки и клей\hfill}`

На с. 148 мы упоминали о том, что в аргументе команды `\vspace` может (вместо длины с `plus`- и/или `minus`-компонентой) стоять команда `\fill` (возможно, с коэффициентом). Как мы теперь понимаем, `\vspace` с таким аргументом также задает бесконечно растяжимый клей. Точнее говоря, `\vspace{\fill}` действует так же, как `\hfill`, в то время как команда `\vspace{0.3\fill}` задает клей, растяжимость которого составляет 30% от растяжимости `\hfill` (тем не менее, этот клей также «бесконечно растяжим» в том отношении, что его присутствие отменяет растяжимость `\hfil`'ов и клея с конечными `plus`-компонентами).

3.4. Бесконечно сжимаемые интервалы

Мы уже два раза упомянули про клей с бесконечной сжимаемостью. Настало время объяснить, какими командами его создавать. Из многих способов укажем один, наиболее часто встречающийся. Команда `\hss` вставляет в строку клей, естественный размер которого равен нулю, и который при этом обладает бесконечной растяжимостью (подобно `\hfil`) и бесконечной сжимаемостью. Типичное применение такого «бесконечно сжимаемого» клея — создавать блоки, ширина которых меньше реального размера текста, или блоки с наложением текстов. В самом деле, посмотрите на такой пример:

Кот Пес	`\hbox to 50pt {Кот\hss Пес}`
КотПес	`\hbox{Кот\hss Пес}`
КоПес	`\hbox to 30pt {Кот\hss Пес}`
ПКеост	`\hbox to 15pt {Кот\hss Пес}`
ПесКот	`\hbox to 0pt {Кот\hss Пес}`

Если мы просим сделать ширину блока больше естественной, команда `\hss` действует так же, как и `\hfil`; когда мы создаем блок с естественной шириной, слова «Кот» и «Пес» стоят вплотную друг к другу (естественная ширина клея, созданного `\hss`, равна нулю). Интересные вещи начинаются, когда мы просим, чтобы ширина была 30 pt (что меньше естественной). Интервал между словами при этом приходится уменьшить; поскольку его естественный размер равен нулю, то после уменьшения интервал становится отрицательным, т. е. слово «пес» сдвигается влево (накладываясь на слово «Кот»), причем сдвигается так, чтобы ширина блока (т. е. расстояние от начала слова «Кот» до конца слова «Пес») равнялась требуемым 30 pt. Когда же мы наконец просим, чтобы ширина блока равнялась нулю, слову «Пес» приходится сдвинуться влево настолько, чтобы расстояние от его конца до начала слова «Кот» равнялось нулю — иными словами, кот и пес меняются местами! Заметим, кстати, что точка отсчета всех наших блоков совпадает с точкой отсчета буквы К из слова «Кот».

Еще один пример использования `\hss`: как создать блок, точка отсчета которого будет находиться в правом, а не левом конце текста (мы столкнулись с этой проблемой в гл. V — см. с. 198)? Ответ: надо сказать

`\hbox to 0pt{\hss` *текст*`}`

и все будет в порядке. В самом деле, *текст* имеет ширину, отличную от нуля; чтобы блок имел в итоге нулевую ширину, приходится «уменьшать» тот интервал, где стоит `\hss`; так как интервал уже нулевой, то это уменьшение сводится к тому, что текст сдвигается влево до тех пор, пока расстояние между его концом и точкой отсчета не станет равным нулю — а это и означает, что правый конец текста стал его точкой отсчета. В гл. V мы сказали, что эта проблема решается с помощью TeX'овской команды `\llap`, а теперь мы видим, как ее можно определить:

```
\newcommand{\llap}[1]{\hbox to 0pt{\hss #1}}
```

Кстати говоря, именно так она и определяется.

А если сказать

`\hbox to 0pt{`*текст*`\hss}`

то что, спрашивается, будет? Ответ: на сей раз будет уменьшаться интервал *после* текста; стало быть, сам текст никуда не сдвинется, но после него будет сделан такой «отрицательный пробел», чтобы суммарная ширина равнялась нулю. Иными словами, TeX будет просто считать, что ширина блока равняется нулю — мы обманули TeX, убедив его, что наш текст не занимает места по горизонтали! Для такого обмана (к нему приходится прибегать нередко) предусмотрена специальная TeX'овская команда `\rlap`, определяемая так:

```
\newcommand{\rlap}[1]{\hbox to 0pt{#1\hss}}
```

Все это также напоминает ситуацию с командой `\lefteqn` — и напоминает не случайно, поскольку эта команда определяется фактически так:

```
\newcommand{\lefteqn}[1]{\rlap{$\displaystyle #1$\hss}}
```

3.5. Еще раз о линейках

В аргументе команды `\hbox` может присутствовать и TEX'овская команда `\vrule`. Ее ценность в том, что она автоматически создает линейку, высота и глубина которой равна высоте и глубине объемлющего блока (ширина этой линейки будет по умолчанию равна 0,4 пункта). Как объяснялось в разд. III.10, можно задать в явном виде ширину линейки с помощью ключевого слова `width`, высоту — с помощью ключевого слова `height`, а также (о чем в гл. III не говорилось) глубину с помощью ключевого слова `depth` (эти три ключевых слова могут следовать после `\vrule` в произвольном порядке). Приведем один пример использования `\vrule` внутри `\hbox`.

| Иногда используется следующий способ выделения текста: абзац набирается с некоторым отступом от левого поля, а слева от него, вровень с левым полем, печатается вертикальная линейка.

Предыдущий абзац в исходном тексте выглядел так:

```
\begin{flushleft}
\hbox{%
\vrule\hspace{.5em}\parbox{.9\textwidth}%
{Иногда используется следующий способ выделения текста:
абзац набирается с некоторым отступом от левого поля,
а слева от него, вровень с левым полем, печатается
вертикальная линейка.}}
\end{flushleft}
```

Этот текст нуждается в некоторых пояснениях. Во-первых, в последней строке первая из фигурных скобок закрывает аргумент команды `\parbox`, а вторая — `\hbox`. Во-вторых, мы воспользовались окружением `\flushleft`, чтобы LaTeX сам позаботился о разумных отступах до и после абзаца. Параметр `\textwidth` означает, как мы помним, ширину страницы. Теперь рассмотрим, что присутствует внутри `\hbox`. Сначала там идет линейка, затем отступ на 0.5 em, и затем — огромная «буква», созданная командой `\parbox`. Согласно общему правилу, высота и глубина линейки, заданной командой `\vrule`, равна высоте и глубине объемлющего блока, а они в нашем случае совпадают с высотой и глубиной «огромной буквы» (ведь кроме нее, другого текста в нашем `\hbox` нет). Тем самым линейка получается как раз нужных размеров, что и требовалось!

Обратите еще внимание на знак процента после `\hbox{` — без него получилось бы, что аргумент команды `\hbox` начинается с пробела, соответственно и линейка начиналась бы не с начала, а после пробела (ср. с. 18).

На самом деле в предыдущем примере было бы лучше, если бы правый край выделенного абзаца шел вровень с правым краем остального текста. Чтобы добиться этого, надо первый аргумент команды `\parbox` не взять с потолка, а вычислить. Для этого нам понадобятся переменные со значением длины. Предполагая, что мы определили с помощью `\newlength` переменные `\shirina` и `\raznost`, сделаем вот что:

```
\begin{flushleft}
\shirina=\textwidth
\settowidth{\raznost}{\vrule\hspace{.5em}}
\addtolength{\shirina}{-\raznost}
\noindent\hbox{%
\vrule\hspace{.5em}\parbox{\shirina}%
{Иногда используется ...
... линейка.}}
\end{flushleft}
```

Мы воспользовались командой `\settowidth`, чтобы найти размер, который занимает линейка вместе с пробелом. Кстати, если просто написать `\hbox{\vrule\hspace{.5em}}`, то на печати мы ничего не увидим (внутри `\hbox`'а никакого текста нет, так что высота и глубина линейки равна нулю и она тем самым невидима); однако же эта команда создаст пробел, величина которого равна `0.4pt` плюс `0.5em`. Заключительное замечание: поскольку `flushright`, как и всякое окружение, ограничивает группу, все наши манипуляции с параметрами `\shirina` и `\raznost` забудутся по выходе из этого окружения.

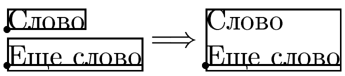
4. Команда `\vbox`

Теперь рассмотрим вторую основную команду TeX'а для генерации блоков — команду `\vbox`. Эта команда создает блок, обрабатывая текст в *вертикальном* режиме. Вот первый пример:

Слово
Еще слово

```
\vbox{\hbox{Слово}
\hbox{Еще слово}}
```

Получаемый блок имеет вид:



Как видите, блоки, создаваемые `\hbox`, ставятся один под другим таким образом, чтобы их точки отсчета лежали на одной вертикальной прямой.

Прежде чем идти дальше, обсудим, что́ может содержаться в аргументе команды `\vbox`. Там могут присутствовать любые TEX'овские команды, допустимые между абзацами (т. е. в вертикальном режиме): команды `\vspace`, команды смены шрифта, присваивания значений различным параметрам, команды `\newcommand` и `\renewcommand` и т. п. Что же касается команд, которым соответствует что-либо на печати, то будем считать, что из них в аргументе `\vbox` возможны только TEX'овские команды `\hbox`, `\vbox` и `\hrule`, а также `\copy`, о которой речь пойдет позже. В частности, недопустим ни текст, ни LATEX'овские команды `\mbox`, `\parbox`, `\rule` и т. п. Если вам требуется воспользоваться возможностями таких команд, «прячьте» их в `\hbox`, например, так:

```
\hbox{\raisebox{1pt}[2em][3em]{...}}
```

На самом деле в аргументе команды `\vbox` может находиться и обычный текст; при появлении первой же буквы или, скажем, команды `\mbox` или другой команды LATEX'а для генерации блоков TEX переходит в горизонтальный режим, который продолжается до команды, завершающей абзац (`\par` или пустой строки). Мы не будем вдаваться в детали; для тех приложений, которые мы имеем в виду, достаточно использовать команду `\vbox` так, как было предписано выше.

Когда TEX при выполнении команды `\vbox` составляет блоки друг с другом, он располагает их так, чтобы их базисные линии были, по возможности, на равных расстояниях друг от друга, так что обычно между блоками будет присутствовать дополнительный пробел. С другой стороны, линейки, созданные командой `\hrule`, приставляются к блокам без дополнительного пробела. Чтобы при этом линейка не оказалась вплотную к тексту, удобно в соответствующий блок вставить `\strut`. Следующий пример призван пояснить сказанное:

Неудачно:
Два слова

Лучше так:
Два слова

```
Неудачно:\\
\vbox{\hbox{Два слова}
      \hrule}

Лучше так:\\
\vbox{\hbox{\strut Два слова}
     \hrule}
```

Как обычно, `\vbox` посреди абзаца ведет себя просто как большая буква. Обратите также внимание, что мы не пытались убрать лишний пробел между `\hbox` и `\hrule`: в вертикальном режиме пробелы никакого влияния на текст не оказывают.

Вот еще пример, когда с помощью комбинации блоков и линеек текст берется в рамку:

$\boxed{\text{Текст в рамке}}$

```
\vbox{\hrule
\hbox{\vrule\,\strut
Текст в рамке\,\vrule}
\hrule}
```

По-прежнему мы используем `\strut`, чтобы горизонтальные линейки не подходили слишком близко к тексту (и `\,` для той же цели по горизонтали).

5. Блоковые переменные

В тех случаях, когда один и тот же фрагмент текста (например, фрагмент псевдорисунка, являющийся аргументом команды `\multiput`) используется многократно, бывает полезно сверстать этот фрагмент раз и навсегда, а затем просто повторять его: это сэкономит как количество нажатий на клавиши, так и машинное время. Использование макроопределения в данном случае времени не сэкономит: если мы напишем, например,

```
\newcommand{\abcd}{\parbox{6cm}%
{Когда в товарищах согласья нет,
на лад их дело не пойдет, и выйдет
из него не дело~--- только м\'ука.}}
```

то при каждой обработке команды `\abcd` это макроопределение будет заново развертываться, и TEX будет заново находить переносы и места разрыва строк в одном и том же отрывке из «Лебедя, рака и щуки». Чтобы не заставлять TEX много раз повторять идентичные операции по верстке текста, надо сделать так. Во-первых, определим «блоковую переменную», которая будет хранить сверстанный фрагмент текста. Это делается с помощью команды `\newsavebox`. Единственный аргумент этой команды — имя новой блоковой переменной, которое должно удовлетворять тем же условиям, что любые имена TEX'овских команд: либо backslash с одной не-буквой, либо backslash с последовательностью букв. Имя новой блоковой переменной не должно совпадать с именем уже существующей команды или переменной длины (если вы попытаетесь нарушить это правило, LATEX выдаст сообщение об ошибке). Во-вторых, присвоим нашей блоковой переменной значение — блок, и будем в дальнейшем этот блок использовать.

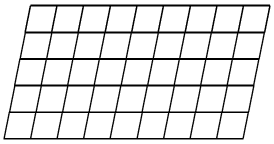
Давайте приведем пример того, как можно этим пользоваться. На с. 203 мы приводили пример псевдорисунка — наклонной решетки, и там же мы отметили, что экономнее было бы создать наклонный отрезок раз и навсегда, а затем только повторять его. Теперь мы можем объяснить, как это сделать. Создадим блоковую переменную под названием `\blok`, написав в преамбуле следующее:

```
\newsavebox{\blok}
```

Теперь сверстаем тот текст, который будет храниться в нашей блоковой переменной, и запишем его в эту переменную. Для этих целей используется команда `\sbox` с двумя обязательными аргументами: первый — имя блоковой переменной, второй — текст, который в нее записывается. Итак:

```
\sbox{\blok}{\line(1,5){10}}
```

А теперь можно воспользоваться нашей блоковой переменной. Чтобы напечатать содержимое блоковой переменной, используется команда `\usebox` с одним обязательным аргументом — именем переменной. В нашем случае мы используем блоковую переменную в аргументе команды `\multiput`:



```
\begin{picture}(100,50)
\multiput(0,0)(10,0){10}%
{\usebox{\blok}}
\multiput(0,0)(2,10){6}%
{\line(1,0){90}}
\end{picture}
```

Можно было бы сделать аналогичный трюк и с горизонтальными линейками решетки, но большой экономии это не даст: горизонтальные и вертикальные линейки на псевдорисунках LaTeX не собирает из отдельных символов, а создает «в один присест» с помощью команд `\hrule` и `\vrule`, что и так достаточно быстро.

Текст, присутствующий в аргументе команды `\sbox`, будет сверстан в виде блока так, как если бы этот текст был передан в качестве аргумента команде `\hbox` или `\mbox`. Тем самым в аргументе `\sbox` может быть все то же, что может присутствовать в аргументе `\hbox` или `\mbox`. Если команда `\sbox` была дана внутри группы, то по выходе из этой группы содержимое блоковой переменной забудется.

Наряду с командой `\sbox` есть еще и команда `\savebox`, относящаяся к ней примерно так же, как `\makebox` относится к `\mbox`: между первым и вторым обязательным аргументами команды `\savebox` могут присутствовать необязательные аргументы, имеющие тот же смысл и записывающиеся так же, как необязательные аргументы команды `\makebox`. Например,

```
\savebox{\пример}[4cm][r]{Слово}
```

даст тот же результат, что и

```
\sbox{\пример}{\makebox[4cm][r]{Слово}}
```

Наряду с LaTeX'овской командой `\usebox` есть похожая на нее, но не идентичная, TeX'овская команда `\copy`. Используется она так:

Однажды Лебедь, Рак и Щука...	`\sbox{\blok}{Рак}` `Однажды Лебедь,` `\copy\blok{} и Щука\ldots`

Обратите внимание, что при использовании команды `\copy` имя блоковой переменной *не* заключается в фигурные скобки! Различие между `\copy` и `\usebox` такое же, как между `\hbox` и `\mbox`: будучи употребленными внутри абзаца (или, скажем, в аргументе команд `\put`, `\hbox` или `\mbox`), эти две команды действуют совершенно одинаково, а вот будучи употребленным между абзацами, LaTeX'овское `\usebox` начинает новый абзац, в то время как TeX'овское `\copy` просто подверстывает блок к странице, нового абзаца не начиная. Эту разницу следует иметь в виду, когда вы работаете с командой `\leaders`: выгоднее сверстать блок один раз и записать его в блоковую переменную, а затем в команде `\leaders` писать просто `\copy`. Пример:

$*$ $*$ $*$ $*$ $*$ $*$	`\savebox{\blok}[1cm]{$*$}` `\hbox to \textwidth` `{\leaders\copy\blok\hfil}`

В этой ситуации по TeXническим причинам сказать `\usebox` нельзя.

Для тех, кто будет читать следующую главу, скажем еще об одной конструкции, связанной с блоковыми переменными. Именно, если `\blok` — блоковая переменная, то можно «измерить» ширину, высоту и глубину блока, записанного в этой переменной, с помощью TeX'овских команд `\wd`, `\ht` и `\dp`. Точнее говоря, сочетания `\wd\blok`, `\ht\blok` и `\dp\blok` можно использовать в точности так же, как TeX'овские параметры со значением длины, значения которых равны ширине, высоте и глубине блока:

12345 345 45	`\sbox{\blok}{12345}\copy\blok` `\hbox to \wd\blok{\hfil 345}` `\hbox to \wd\blok{\hfil 45}`

Для большинства элементарных приложений в LaTeX'е вполне хватает возможностей измерения блоков, предоставляемых командой `\settowidth` и ее аналогами, но в главе IX нам встретятся ситуации, в которых без `\wd` не обойтись.

Глава IX

Модификация стандартных классов

Мне кажется, гиппопотама
Зовут так длинно для того,
Чтоб сторож из глубокой ямы
Пореже вызывал его.

С. Я. Маршак

Эта глава предназначена для тех, кого не удовлетворяет оформление, предлагаемое стандартными классами LaTeX'а. Возможно, вам даже захочется создать свой собственный класс документов вместо стандартных `article`, `proc`, `report` или `book`. Задача эта выполнимая, но для этого надо в деталях знать, во-первых, книгу [2], а во-вторых — исходные тексты LaTeX'а (они доступны). У читателя настоящей книги таких познаний не предполагается, так что мы предлагаем нечто более скромное: научиться модифицировать оформление одного конкретного документа.

Прежде чем двигаться дальше — два предупреждения. В этой главе мы расскажем вам, как можно довольно сильно изменить стандартное LaTeX'овское оформление: вы научитесь менять по своему усмотрению шрифты в заголовках, интервалы, отделяющие заголовки от текста, и много других подобных вещей. Но вот первое предупреждение: если вы не являетесь профессиональным полиграфистом, применяйте эти познания с осторожностью. Не пытайтесь менять сразу много разных элементов оформления или резко изменять какие-то параметры: лучше осторожно менять только то, что вам действительно нужно. Учтите: когда дилетант берется за оформление книги, в девяти случаях из десяти результат бывает достоин сожаления.

Второе предупреждение: стиль оформления записан в специальных «стилевых» или «классовых» файлах, входящих в комплект поставки

LaTeX'а. Ни в коем случае не меняйте ничего в этих файлах: все изменения в стиле оформления надо записывать в собственном стилевом пакете, как объяснено ниже.

1. С чего начать

Кое-какие изменения в оформлении документа вы делать уже умеете: например, в гл. IV рассказывалось, как можно, присвоив в преамбуле новые значения нескольким параметрам, изменить размер полей или текста. Однако для более серьезной модификации оформления приходится иметь дело со специальными командами LaTeX'а, содержащими в своем имени символ `@`. Поскольку `@` — не буква, просто так до этих команд не добраться[1]. Поэтому действовать нужно следующим образом.

Если вы хотите серьезно менять стандартное оформление, нужно создать свой собственный стилевой пакет. Пусть вы решили, что он будет называться `mystyle`. Тогда надо создать файл под названием `mystyle.sty` и начать документ так (подразумевается, что вы хотите печатать шрифтом кегля 11 и отталкиваетесь от класса `book`; в других случаях — с очевидными изменениями):

```
\documentclass[11pt]{book}
\usepackage{mystyle}
```

После `\usepackage` можно писать сразу `\begin{document}`; все установки параметров, определения макросов, и т. п. лучше делать уже не в преамбуле, а непосредственно в файле `mystyle.sty` (чтобы не запутаться, устанавливая один и тот же параметр по-разному в двух разных местах).

Для оформления документа вам, скорее всего, понадобятся какие-нибудь уже существующие стилевые пакеты (если в тексте много формул, то вы захотите подключить пакет `amsmath`, если есть таблицы, то пакет `array`,...). Начать свой личный стилевой пакет надо с того, что подключить эти пакеты. При этом нужно использовать команду `\RequirePackage` (вместо знакомой вам `\usepackage`), например, так:

```
\RequirePackage{array,longtable}
\RequirePackage[noamsfonts]{amsmath}
```

(необязательный аргумент команды `\RequirePackage` означает то же самое и используется так же, как у команды `\usepackage`).

[1] Если просто написать `\@addtoreset` (скоро вы узнаете, что это значит), то TeX воспримет это как команду `\@` (имя — из одной не-буквы!), за которой следует текст `addtoreset`.

На крайний случай, если вам понадобилось использовать команду с символом `@` в имени не в стилевом пакете, а прямо в тексте документа, предусмотрены команды `\makeatletter` и `\makeatother`. Первая из них делает `@` буквой, а вторая восстанавливает status quo. Если вы использовали в тексте `\makeatletter`, не забудьте написать и `\makeatother` сразу после текста, в котором использовалась `@` в имени команд.

Итак, стандартные стилевые пакеты загружены. После этого надо записать в свой стилевой пакет ваши личные команды для модификации оформления. Начнем.

2. Снова о счетчиках

Для начала расскажем о некоторых манипуляциях со счетчиками, которые иногда бывают полезны. До сих пор мы обходили их молчанием, поскольку команды, используемые для этих манипуляций, содержат `@` в своих именах.

2.1. Как подчинить один счетчик другому

Первый из приемов, о которых пойдет речь, связан с отношением подчинения между счетчиками. Мы знаем, что при создании счетчика с помощью команды `\newcounter` можно задать и счетчик, которому он будет «подчинен» (см. с. 248 и ниже). Но как быть, если уже существует никому не подчиненный счетчик, а мы хотим его кому-то подчинить? Например, за нумерацию сносок отвечает счетчик `footnote`; в стиле `article` этот счетчик определяется как никому не подчиненный, благодаря чему нумерация сносок выходит сплошной в пределах всего документа. Если мы хотим, чтоб нумерация сносок начиналась заново в каждом разделе, то можно в стилевом пакете написать так:

```
\@addtoreset{footnote}{section}
```

Первый аргумент команды `\@addtoreset` — имя подчиняемого счетчика, второй — имя подчиняющего.

Разумеется, для того чтобы осмысленно применять описанную команду, надо знать, какие счетчики определены в стандартных стилях и кому они подчинены (или не подчинены). Эта информация содержится в конце раздела.

При ознакомлении с командой `\@addtoreset` может возникнуть искушение написать

```
\@addtoreset{footnote}{page}
```

чтобы сноски нумеровались заново на каждой странице. К сожалению, по TeXническим причинам это может не дать желаемого результата: если сноски оказываются на нескольких страницах подряд, то может случиться так, что на второй из этих страниц нумерация сносок начнется не с 1.

Типичный случай использования команды `\@addtoreset` возникает, если класс документа — `article`. В этом случае часто пишут

```
\@addtoreset{equation}{section}
```

чтобы нумерация уравнений была не сплошной, как предусмотрено стандартом, а начиналась заново в каждом разделе. Разумеется, в этом случае надо будет переопределить команду `\theequation`.

Если вы подключили пакет `amsmath`, то эту операцию можно осуществить и попросту в преамбуле документа: команда `\numberwithin`, принимающая в точности те же аргументы, что и `\@addtoreset`, осуществляет подчинение счетчика и к тому же переопределяет соответствующим образом `the`-команду.

2.2. Ссылочный префикс

Вторая тонкость, о которой пойдет речь, связана с автоматическими ссылками и `the`-командами. Предположим, что перед исполнением команды `\label{metka}` последним счетчиком, подвергшимся увеличению с помощью `\refstepcounter`, был `abcd`. В гл. VII мы говорили, что при этом команда `\ref{metka}` представит на печати значение этого счетчика «в соответствии с командой `\theabcd`». Настало время сознаться, что это полуправда. На самом деле команда `\ref` напечатает перед `\theabcd` еще и так называемый «ссылочный префикс» счетчика. Содержимое ссылочного префикса к счетчику `abcd` записано в команде `\p@abcd` (буква `p`, конечно, латинская). В момент создания счетчика эта команда определяется как макрос с «пустым» замещающим текстом, так что у таких счетчиков, как `chapter` или `section` в стандартных классах, ее следов не видно. Можно, однако, переопределить эту команду, чтобы ссылочный префикс реально печатался. Вот пример работы со ссылочным префиксом.

В классе `book` вид номера главы и номера раздела определяется следующим образом: команда `\thechapter` определена стандартным образом как `\arabic{chapter}` (такое определение, как мы помним, автоматически производится при создании счетчика), а внешний вид номера раздела определен как

```
\renewcommand{\thesection}{\thechapter.\arabic{section}}
```

Из-за этого номер третьего раздела второй главы печатается в заголовке как 2.3. Пусть мы хотим, чтобы номера разделов в заголовках не содержали номера главы; тогда можно написать

```
\renewcommand{\thesection}{\arabic{section}}
```

но при этом возникнет другая неприятность. Автоматические ссылки на номер раздела, генерируемые командой `\ref`, теперь дадут на печати одно и то же число 3 как для третьего раздела второй главы, так и для третьего раздела четвертой главы: ведь из команды `\thesection` информация о номере главы ушла! Чтобы справиться с этой неприятностью, поместим утерянную информацию в ссылочный префикс счетчика `section`:

```
\renewcommand{\p@section}{\thechapter.}
```

Теперь будет печататься 2.3 при ссылке на третий раздел второй главы и 4.3 при ссылке на третий раздел четвертой главы: хотя `\thesection` в обоих случаях дает просто 3, печатающийся перед ним `\p@section` обеспечивает печать номера главы и точки. Кстати говоря, именно такой прием применен при подготовке книги, которую вы читаете.

2.3. Русский аналог `\alph`

В разд. VII.3 мы отмечали, что было бы желательно иметь команду, аналогичную `\alph` или `\Alph`, печатающую русскую букву с номером, равным значению счетчика. Сейчас мы расскажем, как вы ее можете определить в своем стилевом пакете. К сожалению, в этом определении используются не рассматривавшиеся нами средства TeX'а, так что рецепт вам придется воспринять сугубо догматически.

Итак, пусть вам нужна команда `\ralph`, принимающая в качестве аргумента имя счетчика и дающая на печати русскую букву, чей номер в алфавите совпадает со значением счетчика; если этот номер неположителен или превышает число букв в алфавите, команда будет (как и команда `\alph` в аналогичном случае) выдавать сообщение об ошибке. Определить эту команду (в вашем стилевом пакете) можно так:

```
\newcommand*{\ralph}[1]{\@ralph{\@nameuse{c@#1}}}
\newcommand*{\@ralph}[1]%
{\ifcase #1\or а\or б\or в\or г\or д\or е\or ж\or з\or и\or
к\or л\or м\or н\or о\or п\or р\or с\or т\or у\or
ф\or х\or ц\or ч\or ш\or щ\or э\or ю\or
я\else\@ctrerr \fi}
```

(как иногда делают, мы пропускаем буквы ё, й, ъ, ы, ь). Аналогичным образом можно определить команду `\Ralph`, печатающую прописную букву (в этом случае «вспомогательную команду» с символом `@` в имени надо тоже назвать как-нибудь по-другому, скажем, `\@Ralph`). Если вы будете модифицировать это определение, не сделайте в нем пробела между буквой и `\or` или `\else` (конец строки — это тоже пробел), иначе ваша команда будет печатать лишний пробел.

2.4. Кто кому подчинен в стандарте

Нам осталось выполнить свое обещание и рассказать, какие счетчики определены в LaTeX'овском стандарте и каковы отношения подчинения между ними.

В классах `article` и `proc` счетчик `section` определен как

```
\newcounter{section}
```

в то время как в классах `report` и `book`, в которых существуют еще и главы, определяется никому не подчиненный счетчик `chapter` для глав, а счетчик `section` определяется как подчиненный счетчику `chapter`:

```
\newcounter{chapter}
\newcounter{section}[chapter]
```

Остальные счетчики номеров разделов определяются во всех четырех стандартных классах одинаково:

```
\newcounter{part}
\newcounter{subsection}[section]
\newcounter{subsubsection}[subsection]
\newcounter{paragraph}[subsubsection]
\newcounter{subparagraph}[paragraph]
```

Соответствующие этим счетчикам `the`-команды определены в классах `article` и `proc` так:

```
\renewcommand{\thepart}{\Roman{part}}
\renewcommand{\thesection}{\arabic{section}}
\renewcommand{\thesubsection}{\thesection.\arabic{subsection}}
\renewcommand{\thesubsubsection}%
{\thesubsection.\arabic{subsubsection}}
\renewcommand{\theparagraph}%
{\thesubsubsection.\arabic{paragraph}}
\renewcommand{\thesubparagraph}%
{\theparagraph.\arabic{subparagraph}}
```

(мы пишем `\renewcommand`, поскольку все эти `the`-команды уже получили какое-то определение при создании счетчиков).

В классах `report` и `book`, кроме того, определена `the`-команда для счетчика `chapter` и по-другому определена `\thesection`:

```
\renewcommand{\thechapter}{\arabic{chapter}}
\renewcommand{\thesection}%
{\thechapter.\arabic{section}}
```

За нумерацию сносок отвечает счетчик `footnote`. В классах `article` и `proc` этот счетчик определяется как никому не подчиненный:

```
\newcounter{footnote}
```

В классах же `report` и `book` этот счетчик подчинен счетчику `chapter`, так как в них присутствует еще и команда

```
\@addtoreset{footnote}{chapter}
```

В таком же положении, как счетчик `footnote`, находится и отвечающий за нумерацию формул счетчик `equation`: в классах `article` и `proc` он определен как никому не подчиненный, а в классах `report` и `book` он подчинен счетчику `chapter`. Однако же в классах `report` и `book` переопределяется `\theequation`:

```
\renewcommand{\theequation}{\thechapter.\arabic{equation}}
```

Наконец, счетчики `figure` и `table`, отвечающие за нумерацию плавающих иллюстраций и таблиц соответственно, устроены точно так же, как счетчик `equation`: в классах `article` и `proc` они никому не подчинены, а в двух других стандартных классах они подчинены счетчику `chapter` и соответствующие `the`-команды определены как

```
\renewcommand{\thefigure}%
{\thechapter.\arabic{figure}}
```

(аналогично для `table`).

Остались еще счетчики, связанные с нумерованными перечнями. Как объяснялось в разд. VII.3.5, эти счетчики, в зависимости от уровня вложенности `enumerate`, называются `enumi`, `enumii`, `enumiii` и `enumiv`. Все эти счетчики, естественно, последовательно подчинены друг другу, а их ссылочные префиксы определены так (это — единственный случай, когда в стандарте используются нетривиальные ссылочные префиксы):

```
\renewcommand{\p@enumii}{\theenumi}
\renewcommand{\p@enumiii}{\theenumi(\theenumii)}
\renewcommand{\p@enumiv}{\p@enumiii\theenumiii}
```

3. Рубрикация

Здесь мы расскажем о том, как менять оформление разделов документа, предписываемое LaTeX'овским стандартом.

3.1. Что нумеровать и что включать в оглавление

Будет ли раздел документа иметь номер, зависит от двух вещей: «уровня вложенности» раздела и значения счетчика `secnumdepth`. Раздел документа получает номер, если уровень его вложенности меньше или равен значению `secnumdepth` (разумеется, все сказанное относится к случаю, когда LaTeX'овская команда для раздела документа дана без звездочки — иначе нумерации не будет заведомо). Уровни вложенности в стандартных стилях приведены в табл. IX.1. Стало быть, если мы хо-

Таблица IX.1. Уровни вложенности разделов документа

Название раздела	Уровень
`\section`	1
`\subsection`	2
`\subsubsection`	3
`\paragraph`	4
`\subparagraph`	5

тим, чтобы самый мелкий из нумеруемых разделов был `\subsection`, то надо написать в преамбуле документа команду

```
\setcounter{secnumdepth}{2}
```

Что касается команд `\chapter` и `\part`, то соответствующие разделы документа будут нумероваться тогда и только тогда, когда значение `secnumdepth` является неотрицательным числом.

Какие разделы включать в оглавление, определяется другим счетчиком, а именно `tocdepth`: информация о разделе документа вносится в оглавление в том и только том случае, если уровень вложенности раздела меньше или равен значения этого счетчика (и раздел вводится командой без звездочки).

3.2. Модификация команд, задающих разделы

Теперь рассмотрим, что надо делать, чтобы более серьезным образом изменить оформление разделов. Для этого надо переопределить команды `\section`, `\subsection` и т. п., а чтобы научиться их должным образом переопределять, надо узнать, как эти команды определены в стандартных LaTeX'овских классах.

Почти все команды для создания разделов документа определяются в классовых файлах через команду `\@startsection`. Например, команда `\section` определяется (не буквально, а по существу) так:

```
\newcommand{\section}{\@startsection{section}{1}{0pt}%
{-3.5ex plus -1ex minus -.2ex}{2.3ex plus.2ex}%
{\normalfont\Large\bfseries}}
```

В этом определении у команды `\@startsection` указаны шесть аргументов, в которых закодированы различные параметры оформления раздела. Разберем последовательно, что эти аргументы означают.

Первый из аргументов (в нашем случае `section`) — это «внутреннее имя», под которым LaTeX будет узнавать определяемый тип разделов документа. Если вы решили использовать команду `\@startsection` с «нестандартным» первым аргументом (скажем, `abcd`), то заодно придется определить счетчик с именем `abcd`, который будет отвечать за нумерацию разделов, а также команду `\l@abcd`, которая будет отвечать за сбор материала для оглавления (см. подробности в разд. 4), и команду `\abcdmark`, отвечающую за передачу информации для колонтитулов (см. разд. 6),так что без надобности командой `\@startsection` с нестандартным первым аргументом лучше не пользоваться.

Второй аргумент (в нашем случае `1`) — это тот самый «уровень вложенности» раздела, о котором шла речь выше.

Третий аргумент задает отступ заголовка от левого поля (в нашем случае этот отступ равен нулю).

Четвертый аргумент команды `\@startsection` (в нашем случае это `-3.5ex plus -1ex minus -.2ex`) задает величину вертикального отступа, оставляемого перед заголовком. Точнее говоря, вертикальному отступу равен не сам четвертый аргумент, а его абсолютная величина (при определении отступа знаки `-` отбрасываются), а знак `-` означает, что первый абзац нашего раздела будет печататься без абзацного отступа, как и оформляются разделы в стандартных LaTeX'овских стилях. Если задать эти расстояния как положительные числа, то абзацный отступ в первом абзаце подавляться не будет. Тот факт, что отступ перед началом раздела имеет `plus`- и `minus`-компоненты, означает, как водится, что этот пробел обладает растяжимостью и сжимаемостью (см. с. 147).

Пятый аргумент (в нашем случае `2.3ex plus .2ex`) задает величину вертикального отступа после заголовка раздела, а тот факт, что он положителен, означает, что заголовок раздела печатается на отдельной строке (или строках, если в строку он не умещается). Если он будет отрицательным, то заголовок раздела будет печататься не на отдельной строке, а в подбор; значение пятого аргумента `\@startsection` будет означать при этом (после отбрасывания знака минус, естественно) величину дополнительного горизонтального отступа между заголовком раздела и продолжающим его текстом из первого абзаца раздела.

Пятый аргумент `\@startsection` тоже, как видите, может содержать `plus`- и/или `minus`-компоненту.

Наконец, шестой аргумент команды `\@startsection` задает стиль оформления заголовка. Точнее говоря, в этом аргументе записан текст и/или команды, которые будут вставлены перед заголовком раздела. В нашем случае этот аргумент содержит только команды

```
\normalfont\Large\bfseries
```

задающие шрифт, которым заголовок будет напечатан (на последующий текст эта смена шрифта не повлияет, поскольку команды, указанные в шестом аргументе `\@startsection`, будут выполняться внутри группы).

Приведем пример того, как можно изменить стандартное оформление. Пусть нам хочется, чтобы команда `\section` порождала раздел документа, оформленный таким образом:

- номера разделов печатаются римскими цифрами;
- перед номером раздела стоит знак §;
- номер раздела и его заглавие печатаются прямым светлым шрифтом размера `\Large`;
- абзацный отступ в первом абзаце не подавляется.

Как этого добиться? Для начала переопределим команду `\thesection`, определяющую, в каком виде будет представлен на печати номер раздела:

```
\renewcommand{\thesection}{\Roman{section}}
```

А теперь переопределим и саму команду `\section` следующим образом:

```
\renewcommand{\section}{\@startsection{section}{1}%
{\parindent}{3.5ex plus 1ex minus .2ex}%
{2.3ex plus.2ex}{\normalfont\Large\S}}
```

Мы воспользовались `\renewcommand`, поскольку команда `\section` ранее уже была определена. В четвертом аргументе мы убрали знаки `-`, чтобы не подавлять абзацный отступ, а в третьем — задали отступ заголовка от левого поля, равный абзацному отступу (причина этого не программистская, а эстетическая: некрасиво, когда первый абзац раздела идет с отступом, а заголовок начинается вплотную к левому полю). Прочие параметры, задающие размеры отступов, мы оставили такими же, как в стандартной LaTeX'овской команде: они достаточно разумны, и незачем их трогать, если на то нет особых причин.

Как видите, для того, чтобы модифицировать оформление разделов, надо знать размеры стандартных LaTeX'овских параметров оформления, наподобие отступа перед или после заголовком. Для команды `\section` мы привели их выше, а для остальных стандартных команд

Таблица IX.2. Стандартное оформление разделов

	Отступ	Перед	После
`\subsection`	`0pt`	`-3.25ex plus -1ex minus-.2ex`	`1.5ex plus .2ex`
`\subsubsection`	`0pt`	`-3.25ex plus -1ex minus-.2ex`	`1.5ex plus .2ex`
`\paragraph`	`0pt`	`3.25ex plus 1ex minus.2ex`	`-1em`
`\subparagraph`	`\parindent`	`3.25ex plus 1ex minus.2ex`	`-1em`

рубрикации они собраны в табл. IX.2. В ней «отступ» означает отступ заголовка от левого поля, «перед» — отступ перед заголовком (если этот отступ отрицательный, то абзацный отступ в первом абзаце будет подавлен), «после» — отступ после заголовка (если это отрицательное число, то заголовок печатается в подбор). Уровни вложенности разделов см. в табл. IX.1, а что до «внутреннего имени» раздела (первый аргумент команды `\@startsection`), то оно у всех стандартных команд рубрикации совпадает с их именем (`section` для команды `\section` и т. п.). Шестой аргумент команды `\@startsection` содержит в стандартных определениях только команды переключения шрифта; при необходимости заинтересованный читатель воспроизведет его самостоятельно.

Шестой аргумент команды `\@startsection` позволяет автоматически добавлять текст перед номером раздела. К сожалению, не так просто[2] заставить LaTeX автоматически добавлять текст после номера, хотя такая потребность порой возникает (например, хочется, чтобы

[2]Но возможно: это сделано, например, в пакете `russcorr`, использованном при подготовке данной книги. См. приложения Е и Ж.

в заголовках разделов после номеров стояли точки, чего стандартное LaTeX'овское оформление не предусматривает).

Можно заставить LaTeX ставить точки после номеров, написав

```
\renewcommand{\thesection}{\arabic{section}.}
```

но после этого автоматические ссылки на раздел, сгенерированные с помощью команды `\ref`, также будут заканчиваться точками, что нелепо.

Оформление глав отличается от оформления остальных разделов тем, что слово «Глава» и номер главы печатаются на отдельной строке. С помощью команды`\@startsection` определить такой раздел нельзя, поэтому главы определяются в LaTeX'овских классах иначе. Не будем вдаваться в подробности, как именно, а вместо этого рассмотрим единственно важный для нас вопрос: как можно менять оформление глав.

Бóльшая часть оформления главы задается в определении команды `\@makechapterhead`, так что для модификации оформления именно ее и надо переопределять. Рассмотрим, как `\@makechapterhead` определяется в стандарте. Этой команде передается один аргумент — заголовок главы. В переводе с TeX'а на LaTeX определение выглядит так (не забудьте, что `#1` — это аргумент, т. е. текст заголовка):

```
\newcommand{\@makechapterhead}[1]{% Начало макроопределения
  \vspace*{50 pt}% Пустое место вверху страницы
  {\parindent=0pt
   \raggedright \normalfont\huge\bfseries
   \@chapapp{} % \@chapapp печатает слово "Глава" (см. ниже)
    \thechapter \par % номер главы - в отдельной строке
    \vspace{20 pt} % между словом "Глава" и ее заголовком
    \normalfont\Huge\bfseries #1\par % заголовок главы
    \nopagebreak     % чтоб не оторвать заголовок от текста
    \vspace{40 pt}   % между заголовком и текстом
  }% конец группы.
}% конец макроопределения
```

Разберем эту «программу». Первая команда `\vspace*` оставляет пустое место вверху страницы (поскольку главы начинаются с новой страницы). Далее печатается слово «Глава», ее номер и заголовок главы; поскольку заголовок может не поместиться в строку, надо предусмотреть, какие будут при этом параметры верстки абзаца. Как видите, устанавливается нулевое значение абзацного отступа и отсутствие выравнивания по правому краю; чтобы такой режим не распространился на дальнейший текст, соответствующие команды даны внутри группы. Внутри этой же группы определен шрифт, которым будет печататься заголовок.

Команда `\@chapapp` печатает слово «Chapter», «Глава»,... — одним словом, то, как в вашем документе называются главы. Точнее говоря, по умолчанию эта команда работает так же, как `\chaptername`, а после команды `\appendix` (если таковая есть в вашем файле) начинает работать как `\appendixname` (см. с. 170). Если в вашем тексте все главы называются одинаково, то при переопределении `\@makechapterhead` можно не мудрить, а прямо заменить `\@chapapp` на `Глава`. Не забудьте только оставить пробел между этим словом и номером главы (выше это было сделано с помощью обычного трюка с парой фигурных скобок).

Остальное в приведенном выше определении разъяснений, надо думать, не требует. Скорее всего, вы захотите в этой команде изменить шрифт, которым печатается заголовок главы, или же интервалы, отделяющие заголовок от остального текста. Чтобы изменить вид, в котором представляется на печати номер главы, надо, как водится, переопределить команду `\thechapter`. Можно при желании задать и какое-нибудь более сложное оформление заголовка с помощью блоков и линеек — все зависит от вашей фантазии и вкуса!

Надо еще сказать, что мы немного обманули читателя: на самом деле в стандартных классах команда `\@makechapterhead` определена таким образом, что если значение счетчика `secnumdepth` отрицательно, то команды, записанные в строках с пятой по седьмую, не исполняются (и номер главы не печатается). В нашем определении эти команды будут исполняться всегда, вне зависимости от значения `secnumdepth`; если вы переопределяете `\@makechapterhead` и не хотите, чтобы главы нумеровались, просто удалите соответствующие строки из определения.

Кроме того, если в классе `book` вы переопределите `\@makechapterhead`, отталкиваясь от этого определения, то главы, заданные как `\chapter` без звездочки, будут нумероваться, невзирая на команды `\frontmatter` и `\backmatter` (но вы ведь сможете поставить две–три звездочки самостоятельно?).

За оформление заголовка главы, определенной командой `\chapter` со звездочкой, отвечает команда `\@makeschapterhead`. Ее определение в стандарте аналогично определению `\@makechapterhead`, с тем отличием, что из него удален фрагмент, отвечающий за печать номера:

```
\newcommand{\@makeschapterhead}[1]{%
  \vspace*{50 pt}%
  {\parindent=0pt \raggedright
    \normalfont\Huge\bfseries #1\par
    \nopagebreak
    \vspace{40 pt}}}
```

Кроме оформления заголовка, с оформлением глав можно делать еще две вещи. Во-первых, главы начинаются либо просто с новой страницы (как в классе `report`), либо с новой нечетной страницы (как в классе book); чтобы повлиять на этот выбор, не надо ничего делать в личном стилевом пакете, достаточно просто указать классовую опцию openright или openany (в необязательном аргументе команды \documentclass). Во-вторых, по умолчанию абзацный отступ в первом абзаце главы подавляется; вы можете захотеть сделать так, чтобы он не подавлялся. Чтобы решить эту проблему, надо переопределять уже саму команду \chapter, а для этого надо знать, как она определяется в стандарте. Вот соответствующее определение, опять в переводе на LaTeX с TeX'а, в классе book:

```
\newcommand{\chapter}{\cleardoublepage
   \thispagestyle{plain}%
   \global\@topnum=0
   \@afterindentfalse
   \secdef\@chapter\@schapter}
```

Разбирать это определение мы не будем, чтобы не запутаться в некоторых слишком хитрых TeX'овских и LaTeX'овских конструкциях, а просто скажем две вещи:

- чтобы не подавлялся абзацный отступ в первом абзаце главы, замените \@afterindentfalse на \@afterindenttrue;

- если вы не хотите, чтобы на странице с заглавием главы печаталась колонцифра (номер страницы), замените аргумент команды \thispagestyle с plain на empty.

Что касается команды \part («часть»), то она отличается тем, что заголовок части занимает отдельную страницу. Если вы решили изменить стандартное оформление таких «частей», то проще всего оформить соответствующие две–три страницы вручную.

4. Оглавление, список иллюстраций и прочее

Автоматическая сборка оглавления — многоэтапный процесс. Сначала материал для оглавления (заглавия разделов и номера соответствующих страниц) записывается в специальный файл с тем же именем, что и у основного файла, и расширением toc (в нормальных условиях эта запись обеспечивается командами \chapter, \section и т. д.); при следующем запуске LaTeX'а этот toc-файл считывается (с помощью команды \input), команды, записанные в него, исполняются, и в результате происходит фактическая печать оглавления. Аналогичным образом составляются список иллюстраций и список таблиц (при этом информация записывается в файлы с расширениями `lof` или `lot` соответственно). Давайте научимся влиять на этот процесс.

Сначала расскажем, как составлять оглавление полностью вручную, игнорируя его автоматическую сборку, обеспечиваемую командами типа `\section`. Итак, предположим, что все команды `\section`, `\chapter` и т. п. даны в исходном тексте в варианте со звездочкой, и посмотрим, как можно самому создать оглавление.

Команда `\addtocontents` служит для записи в `toc`- (соответственно, `lof`- или `lot`-) файл любого текста и любых TEX'овских команд. У этой команды два обязательных аргумента. Первый из них должен быть `toc`, `lof` или `lot`, в соответствии с тем, в какой из файлов с оглавлениями вы пишете свой текст. Второй аргумент — то, что вы хотите записать в файл. Если, например, вам взбрело в голову внести в оглавление к вашей книге текст «У попа была собака» (не будем спрашивать, зачем), то можете написать

```
\addtocontents{toc}{У попа была собака\par}
```

(`\par` поставить необходимо, так как до и после выполнения каждой команды, записанной в оглавлении, TEX должен находиться в вертикальном режиме). Если после этого запустить LATEX два раза, то вы увидите в оглавлении свой текст (после первого раза он только попадет в `toc`-файл, а при втором запуске `toc`-файл с этим текстом будет обработан).

С помощью команды `\addtocontents` можно записывать в оглавление не только всякие глупости. Если, например, вы хотите в каком-то месте оглавления провести горизонтальную линейку шириной во всю страницу, то можно написать

```
\addtocontents{toc}{\hrule}
```

и в оглавлении появится линейка. Имейте только в виду, что в аргументе `\addtocontents` необходимо защищать хрупкие команды с помощью команды `\protect` (см. с. 170). В случае с `\hrule` мы обошлись без `\protect`, так как эта команда не хрупка, но если есть сомнения, то лучше команду защитить. Напомним, что `\protect` действует только на непосредственно следующую команду и что команды для смены шрифта или установки пробелов в защите с помощью `\protect` не нуждаются. Приведем пример более разумного применения `\addtocontents`, в котором требуется `\protect`. Пусть вы не хотите, чтобы какая-то из строк в оглавлении начинала новую страницу. Тогда надо перед командой, порождающей эту строку оглавления (обычно таковой будет команда наподобие `\section`), написать в своем файле вот что:

```
\addtocontents{toc}{\protect\nopagebreak}
```

В результате в `toc`-файл запишется команда `\nopagebreak`, и нежелательный разрыв страницы в оглавлении будет предотвращен. Если опустить `\protect`, то получится весьма непонятное сообщение об ошибке.

При совместном использовании команд `\addtocontents` и `\include` возникает следующий неприятный эффект. Пусть ваш файл имеет вид, скажем,

```
\documentclass{book}
\usepackage{mystyle}
\begin{document}
\tableofcontents
\include{ch1}
\addtocontents{\hrule}
\include{ch2}
\end{document}
```

Тогда, вопреки всем ожиданиям, в оглавлении линейка окажется не между записями, отвечающими файлам `ch1.tex` и `ch2.tex`, а после записей, отвечающих файлу `ch2.tex`. Чтобы этого избежать, запишите команду `\addtocontents` в начало файла `ch2.tex` (са́мой первой строчкой).

Чтобы составить полноценное оглавление, надо иметь возможность записать в `toc`- (соответственно, `lof`- или `lot`-) файл не только текст, но и номер той страницы, к которой этот текст относится. Это делается с помощью команды `\addcontentsline`, имеющей такой синтаксис:

`\addcontentsline{`*тип_файла*`}{`*тип_записи*`}{`*текст*`}`

Здесь *тип_файла* — это `toc`, `lof` или `lot`, *текст* — тот текст, который будет записан в оглавление (например, команда `\section` в стандартном стиле `article` в качестве этого текста передает название раздела и его номер; подробности см. ниже). Наконец, *тип_записи* определяет, каким образом будет обрабатываться этот текст при чтении файла с оглавлением. Именно, если второй аргумент в команде `\addcontentsline` был `abcd`, то, когда при следующем запуске LaTeX'а будет читаться `toc`- (соответственно, `lof`- или `lot`-) файл, будет исполнена команда `\l@abcd` с двумя аргументами, первый из которых — текст, записанный в третьем аргументе команды `\addcontentsline`, а второй — номер страницы, на которую попала ваша команда `\addcontentsline`. Например, если в файле было написано

```
\addcontentsline{toc}{abcd}{О слонах}                    (*)
```

и если эта команда попала на страницу 95, то при следующем запуске LaTeX'а в процессе чтения `toc`-файла будет исполняться команда

```
\l@abcd{О слонах}{95}
```

Разумеется, чтобы при этом не получилось сообщения об ошибке, надо, чтобы команда `\l@abcd` была определена. Стало быть, в стилевом пакете должно присутствовать ее определение. Если мы хотим, чтобы запись (∗) в исходном файле порождала в оглавлении строку

О слонах..95

то в преамбуле надо написать вот что:

```
\newcommand{\l@abcd}[2]{\hbox to\textwidth{#1\dotfill #2}}
```

Чтобы при этом страница в оглавлении была указана верно, необходимо команду `\addcontentsline` разместить непосредственно после команды `\section*` (иначе есть опасность, что они попадут на разные страницы).

Если в третьем аргументе команды `\addcontentsline` присутствуют «хрупкие» команды, то их следует, как водится, защитить командой `\protect`; если, с другой стороны, в нем записана `\the`-команда, соответствующая какому-то счетчику, то в `toc`-файл будет записано печатное представление значения этого счетчика по состоянию на тот момент, когда выполнялась `\addcontentsline`. Таким способом можно, например, записать в оглавление номер текущего раздела документа: достаточно сказать

```
\addcontentsline{toc}{abcd}{\thesection. О слонах}
```

Теперь рассмотрим, как именно собирают оглавление стандартные команды наподобие `\chapter` или `\section`. Делают они это также с помощью `\addcontentsline`, при этом ее второй аргумент («тип записи») будет `section` для команды `\section`, `subsection` для команды `\subsection`, — одним словом, «внутреннее имя», под которым LaTeX знает тип разделов документа (напомним, что внутреннее имя передается в качестве первого аргумента команде `\@startsection`). Стало быть, для модификации стиля оформления строк оглавления, соответствующих `\section`, надо переопределять команду `\l@section`, для модификации строк оглавления, соответствующих `\subsection`, надо переопределять `\l@subsection` и т. д. Чтобы было понятно, как их переопределять, рассмотрим, как они определены в стандарте.

В классе `book` команда `\l@section` определена так:

```
\newcommand{\l@section}{\@dottedtocline{1}{1.5em}{2.3em}}
```

Смысл трех выражений, стоящих в фигурных скобках после команды `\@dottedtocline`, таков. Первое выражение — «уровень вложенности» элемента оглавления. Если этот уровень превышает значение счетчика `tocdepth`, то команда `\@dottedtocline` ничего в оглавлении не печатает. Второе выражение — отступ строки оглавления от левого поля. Третье выражение определяет, сколько места в строке оглавления TeX отведет на номер раздела. Результат выглядит на печати так: после отступа, указанного во втором выражении, печатается номер раздела, затем, отступя от *начала* этого номера столько, сколько сказано в третьем выражении, печатается заглавие раздела. Это сделано для того, чтобы заглавия всех разделов печатались в оглавлении одно под другим. После заглавия идут «лидеры» — ряд точек до завершающего строку номера страницы. Если заглавие в строку не укладывается, то оно обычным образом будет перенесено на следующую строку (если есть какие-то неясности, загляните в оглавление к этой книге). Из сказанного следует, что слишком длинный номер раздела может в оглавлении наложиться на заглавие. Средство борьбы с этим — переопределить команду `\l@section` (или `\l@subsection`...), увеличив должным образом второй аргумент команды `\@dottedtocline`.

Параметры оформления элемента оглавления, задаваемого командами, определенными через `\@dottedtocline`, можно менять. Именно, размер места, отводимого на номер страницы, задается значением команды `\@pnumwidth`, которую можно переопределить. В классе `book` эта команда определена как

```
\newcommand{\@pnumwidth}{1.55em}
```

и соответственно на номер страницы отводится `1.55em` места. Если мы хотим, чтобы на номер страницы отводилось `2em`, надо написать

```
\renewcommand{\@pnumwidth}{2em}
```

Еще одна команда, значение которой отвечает за оформление оглавления, — это `\@tocrmarg`. Если запись в оглавлении занимает более одной строки, то значение этой команды задает отступ от правого поля, который будет у всех строк, кроме той последней, что завершается номером страницы. Если вы хотите, чтобы размер этого отступа равнялся `3em`, напишите так:

```
\renewcommand{\@tocrmarg}{3em}
```

Хотя `\@pnumwidth` и `\@tocrmarg` используются для задания размеров, они не являются параметрами со значением длины; запись наподобие `\@tocrmarg=4em` приведет к ошибке!

Наконец, регулировать густоту точек-«лидеров» можно, если переопределить команду `\@dotsep`. В классе `book` она определена как

```
\newcommand{\@dotsep}{4.5}
```

Если вы хотите, чтобы точки шли погуще, попробуйте переопределить ее, заменив `4.5` на число поменьше (число может быть дробным, в нем можно использовать как десятичную запятую, так и десятичную точку):

```
\renewcommand{\@dotsep}{3,9}
```

Напрашивающаяся запись `\@dotsep=3,9` приведет к ошибке.

Команды `\l@subsection` и «более мелкие» определяются в классе `book` так же, как `\l@section`, отличаются только аргументы команды `\@dottedtocline`. Мы собрали значения этих параметров в табл. IX.3.

Таблица IX.3. Стандартные определения `l@`-команд (класс `book`)

	Три аргумента `\@dottedtocline`		
`\l@subsection`	2	3.8em	3.2em
`\l@subsubsection`	3	7.0em	4.1em
`\l@paragraph`	4	10em	5em
`\l@subparagraph`	5	12em	6em

Теперь рассмотрим, как в стандарте определяются записи в оглавлении, соответствующие самым крупным разделам (`\chapter` в классах `book` и `report`, `\section` в двух других классах). Вот (в адаптированном виде, как водится) определение команды `\l@chapter` из стандартного класса `book`. Чтобы было понятно, о чем идет речь, напомним, что в этом определении на место `#1` подставляется информация о номере (если главы нумеруются) и названии главы, а на место `#2` — номер страницы.

```
\newcommand{\l@chapter}[2]%
{\pagebreak[3]
 \vspace{1em plus 1pt}% отбивка перед строкой оглавления
   \@tempdima=1.5em % место для номера главы
   {% Дальнейшее происходит внутри группы...
     \rightskip=\@pnumwidth % см. ниже
     \parfillskip=-\@pnumwidth
     \noindent\bfseries % Начать абзац, установить шрифт
     \addtolength{\leftskip}{\@tempdima}% см. ниже
     \hspace{-\leftskip}#1\nolinebreak
     \hfil\nolinebreak
     \hbox to \@pnumwidth {\hss #2}\par
```

```
    \nopagebreak[3] % лучше бы здесь не рвать страницу...
  }% конец этой группы
}% конец определения
```

Определение, как видите, длинное и сложное, к тому же автору не удалось полностью изгнать из него не упоминавшиеся ранее TEX'овские конструкции. Приведено оно здесь не для того, чтобы вы самостоятельно создавали подобные определения «с нуля», но чтобы вы при необходимости смогли в нем кое-что осторожно изменить. Разберем определение по порядку. В начале встречаются не рассматривавшиеся нами параметры `\rightskip` и `\leftskip`. Эти параметры со значением длины (по умолчанию они равны нулю) имеют следующий смысл: все строки абзаца начинаются с отступом `\leftskip` от левого поля и кончаются с отступом `\rightskip` от правого поля (если речь идет о последней строке, то в дополнение к отступу `\parfillskip`). У нас `\rightskip` устанавливается равным длине, записанной в определении команды `\@pnumwidth` (именно столько места будет отведено на номер страницы), а `\leftskip` устанавливается равным значению переменной `\@tempdima`, определяющей, сколько места будет отведено на номер главы. С другой стороны, параметр `\parfillskip` (см. с. 127) устанавливается равным отрицательной величине `-\@pnumwidth`. Тем самым, если название главы длинное и не поместится в одну строку оглавления, произойдет следующее: все строки, кроме последней, будут заканчиваться на расстоянии `\@pnumwidth` от правого края, а последняя строка, содержащая номер страницы, закончится на расстоянии

$$\texttt{\textbackslash leftskip} + \texttt{\textbackslash parfillskip} = \texttt{\textbackslash @pnumwidth} - \texttt{\textbackslash @pnumwidth} = 0,$$

от края, так что номер страницы будет все-таки прижат вправо.

Между `\@pnumwidth` и `\@tempdima` есть существенная разница. Команда `\@pnumwidth` всегда определяет только место, отводимое на номер страницы, и эту команду можно переопределять в стилевом пакете. С другой стороны, параметр `\@tempdima` используется LaTeX'ом для самых разных целей (в основном — для временного хранения различных длин в процессе каких-то вычислений), и он может измениться в процессе выполнения очень многих LaTeX'овских команд, так что присваивать ему какое-то значение в стилевом пакете совершенно бессмысленно — все равно после этого оно сто раз изменится. Как мог заметить читатель, значение этому параметру присваивается в начале исполнения команды `\l@chapter`, и именно это значение принимается в расчет в дальнейшем. Поэтому, если вы захотите отводить на номер главы, скажем, `2em` вместо `1.5em`, то вам придется переопределить команду `\l@chapter`, заменив третью строку на

```
\@tempdima=2em
```

Нужда в таком переопределении `\l@chapter` возникает, например, если мы переопределяем команду `\thechapter`, чтобы номер печатался римскими цифрами (как в книге, которую вы читаете). Далее, `#1` — это, как уже было сказано, номер и заглавие главы. Точнее говоря, на месте `#1` печатается[3] такой текст (будем считать, что глава называется «Все о слонах»):

```
\makebox[\@tempdima][l]{\thechapter}Все о слонах
```

Таким образом, величина отступа от левого поля до названия главы всегда равна `\@tempdima`; если номер занимает больше места, чем `\@tempdima`, то он наложится на название.

Команда `\hfil` в двенадцатой строке обеспечивает пробел между названием главы и номером страницы. Если вы хотите заполнить этот пробел лидерами, можете в определении `\l@chapter` заменить `\hfil` на, скажем,

```
\leaders\hbox to .5em{\hss.\hss}\hfil
```

(автор не гарантирует, что именно при таком выборе параметров лидеры будут выглядеть красиво).

Наконец, команды `\pagebreak[3]` в начале и `\nopagebreak[3]` в конце играют следующую роль: первая из них призывает LaTeX не печатать, по возможности, строку оглавления, соответствующую главе, внизу страницы, а вторая — не разрывать страницу сразу после строки, посвященной главе (опять же по возможности).

Имейте также в виду, что мы удалили из нашего упрощенного определения `\l@chapter` проверку значения счетчика `tocdepth`, так что если вы в какой-то момент решите не включать главы в оглавление, эту команду надо будет переопределить на «ничего не делать» так:

```
\renewcommand{\l@chapter}[2]{}
```

Команды, определяющие вид записей в списке иллюстраций (соответствующих плавающим иллюстрациям) и списке таблиц (соответствующих плавающим таблицам) называются `\l@figure` и `\l@table` соответственно и определяются в стандарте с помощью `\@dottedtocline`.

До сих пор речь шла о сборке материала для оглавления, списка иллюстраций и т. п. Однако же и у самого оглавления есть заголовок, и его оформление тоже можно менять. Чтобы было понятно, как это делать, опишем, как определена команда `\tableofcontents` в стандартном стиле `article`:

```
\newcommand{\tableofcontents}%
{\section*{\contentsname}\@starttoc{toc}}
```

[3]Начиная с левого поля, невзирая на установленное значение `\leftskip`; именно для этих целей в определение включена команда `\hspace{-\leftskip}`.

Здесь \contentsname — это уже знакомая нам команда, которую при работе с русскими текстами приходится переопределять (см. с. 169). Как видите, заголовок оглавления оформляется просто как заголовок ненумерованного раздела. Вы можете вместо этого оформить заголовок, скажем, с помощью \subsection, или еще каким-либо образом. Новой для вас будет команда \@starttoc. У этой команды предусмотрен один обязательный аргумент. Этим аргументом должен быть toc (для оглавления), либо lot или lof (для списка таблиц или иллюстраций соответственно). Команда \@starttoc читает toc- (соответственно, lot- или lof-) файл и создает оглавление как таковое.

На самом деле в определении \tableofcontents присутствует еще команда, позволяющая задать текст для включения в колонтитулы (вспомним, что \section* сама по себе никакой информации для колонтитулов не дает). Мы не будем здесь вдаваться в скучные подробности. Когда, по прочтении разд. 6, вы научитесь задавать такие команды, вы сможете соответствующим образом переопределить и \tableofcontents.

5. Перечни общего вида

В этом разделе мы завершим рассказ о том, как менять стиль оформления перечней (см. разд. VII.3.5). В гл. III мы назвали перечнями окружения itemize, enumerate и description; помимо этого, к перечням в LaTeX'овском смысле относятся flushleft, flushright, center, quote, quotation, verse, а также окружения, создаваемые с помощью \newtheorem; как мы увидим в разд. 5.3, все эти окружения — частные случаи одной LaTeX'овской конструкции. Поэтому многие параметры оформления этих окружений устанавливаются и модифицируются по единой схеме.

5.1. Отбивки в перечнях

Начнем с важного предупреждения. Чтобы изменить отбивки, с помощью которых оформляются перечни, необходимо, естественно, изменить значение каких-то из перечисляемых в этом разделе параметров. Однако же, если попросту присвоить этим параметрам новые значения в преамбуле или в стилевом пакете, то в большинстве случаев действия это не возымеет. Средства, которые надо применить, чтобы эти изменения подействовали, описаны в следующем подразделе.

Теперь договоримся о терминологии. Каждый перечень LaTeX рассматривает как состоящий из *элементов* (каждый элемент вводится, как мы помним, командой \item). В свою очередь, каждый элемент перечня может состоять из одного или нескольких абзацев. Наконец, у каждого элемента перечня есть свой *заголовок* — «горошина» на первом уровне окружения `itemize`, заданный вами заголовок в окружении `description` и т. п. (У некоторых перечней — например, таковы «теоремы» — перечень состоит из одного-единственного элемента; у таких перечней, как `quote` или `verse`, кроме того, заголовок к этому единственному элементу всегда пуст.)

Вооружившись этими терминами и имея в виду предупреждение, приступим к утомительному перечислению параметров. Все они — параметры со значением длины. Во-первых, параметры `\leftmargin` и `\rightmargin` задают, с каким отступом от левого (правого) поля начинается (заканчивается) текст элементов перечня (полиграфист сказал бы: насколько *втянуты* элементы перечня). Если перечень вложен в другой перечень, то `\leftmargin` и `\rightmargin` обозначают величину втяжки по отношению к объемлющему перечню.

Следующие два параметра влияют на размещение заголовков в перечне. Параметр `\labelsep` задает расстояние между правым краем заголовка и началом текста в элементе перечня, к которому относится этот заголовок, а параметр `\labelwidth` задает место по горизонтали, которое по умолчанию занимает заголовок. Точный смысл этих параметров следующий. При обработке перечня LaTeX сначала пытается поместить заголовок в блок шириной `\labelwidth`. Если места хватает, то именно в такой блок он и помещается, причем выключенным вправо: правый край блока при этом находится на расстоянии `\labelsep` от начала текста, составляющего элемент перечня (так что его левый край будет на расстоянии

`\leftmargin` − `\labelwidth` − `\labelsep`

от левой границы основного текста или объемлющего перечня). Если же ширина заголовка больше, чем `\labelwidth`, то заголовок печатается как есть. Такое, например, регулярно случается при пользовании окружением `description`.

Мы не сказали еще об одном параметре, влияющем на размещение заголовков. Именно, если параметр `\itemindent` отличен от нуля, то каждый заголовок перечня будет дополнительно сдвинут на это расстояние вправо. Соответственно, при определении, на каком расстоянии начинается заголовок элемента перечня, надо будет прибавить значение `\itemindent` к тому, что получается по формуле (∗). По умолчанию значение этого параметра равно нулю.

Если элемент перечня состоит из нескольких абзацев, то по умолчанию во всех этих абзацах абзацный отступ будет отсутствовать. Можно, однако, при желании задать такой режим, что во всех, кроме первого, абзацах каждого элемента перечня будет присутствовать абзацный отступ. Для этого надо задать ненулевую величину этого отступа в параметре `\listparindent`. Кстати, значение этого параметра может быть и отрицательным (в этом случае эффект будет похож на тот, что достигается в обычном тексте установкой параметров `\hangindent` и `\hangafter`).

Параметры, о которых шла речь до сих пор, относились к размещению материала по горизонтали. Теперь займемся «вертикальными» параметрами. Сразу отметим, что все эти параметры являются «растяжимыми» длинами (с. 147), т.е. у них можно задавать `plus`- и `minus`-компоненты.

Первый (и основной) из этих параметров называется `\topsep`. Это величина дополнительного вертикального интервала, который делается перед перечнем и после него (в дополнение к `\parskip` — см. с. 149).

Если перед перечнем оставлена пустая строка (или имеется команда `\par`), то перед и после перечня устанавливается еще и вертикальный отступ, равный `\partopsep` (в дополнение к отступам, заданным параметрами `\parskip` и `\topsep`).

Далее, вертикальный отступ между абзацами внутри одного элемента задается параметром `\parsep` (а не `\parskip`, как в обычном тексте). Между различными же *элементами* перечня, в дополнение к `\parsep`, оставляется еще и вертикальный отступ `\itemsep`. Таким образом, если `\itemsep` отличен от нуля, как это и сделано в стандартных классах, то различные элементы перечня будут более отделены друг от друга, чем абзацы внутри одного элемента перечня.

5.2. Изменение отбивок в перечнях

Теперь настало время объяснить, как именно можно менять вышеописанные параметры. При «входе» в перечень LaTeX в первую очередь вычисляет уровень вложенности перечней: если перечень не вложен ни в какой другой, то этот уровень равен 1, для перечня, вложенного в перечень, уровень равен 2 и т. д. После этого исполняется команда `\@listI`, если уровень равен 1, `\@listii`, если уровень равен 2, и т. д.: имя команды — слово `@list`, к которому добавлен уровень вложенности, записанный римскими цифрами (если уровень вложенности равен 1, то римская цифра записывается прописной буквой `I`, в остальных случаях римские цифры записываются строчными латинскими буквами). В стандартных классах команды `\@listI`, `\@listii` и т. п. определены таким образом, что они в момент входа в перечень устанавливают значение параметров оформления перечня на соответствующем уровне. Именно поэтому переустановка (некоторых из) параметров перечня в стилевом пакете может ничего не дать.

Итак, чтобы менять отбивки в перечнях, надо переопределять команды `\@listI`, `\@listii`, ..., `\@listvi`. Видимо, самое практичное — переопределить в своем стилевом пакете команды `\@listI`, `\@listii`, ..., `\@listvi` (или не все из них — в зависимости от того, какова реальная максимальная глубина вложенности перечней в вашем документе), присвоив всем перечисленным в предыдущем разделе десяти параметрам явные значения. Например, можно написать что-нибудь в таком роде:

```
\renewcommand{\@listI}{%
\leftmargin=25pt
\rightmargin=0pt
\labelsep=5pt
\labelwidth=20pt
\itemindent=0pt
\listparindent=0pt
\topsep=8pt plus 2pt minus 4pt
\partopsep=2pt plus 1pt minus 1pt
\parsep=0pt plus 1pt
\itemsep=\parsep}
```

Здесь всем параметрам оставлены стандартные значения, за исключением `\parsep` и `\itemsep`: в отличие от положительных значений, задаваемых в стандарте, мы устанавливаем их такими же, как обычная вертикальная отбивка между абзацами (как вы помните, она обозначается `\parskip` и обладает нулевым естественным размером и растяжимостью `1pt`). На взгляд автора, такие перечни смотрятся лучше.

Если какой-то из параметров (например, `\labelsep`) одинаков в перечнях всех уровней, то достаточно прописать его установку только в определении `\@listI`: при входе в перечень верхнего уровня этот параметр будет установлен командой `\@listI`, и LaTeX будет его помнить, пока не выйдет из «наружного» перечня.

5.3. Окружения `list` и `trivlist`

Все LaTeX'овские перечни являются на самом деле частными случаями одной общей конструкции — окружения `list`. Рассмотрим, как это окружение работает.

Окружение `list` имеет два обязательных аргумента. Общий вид этого окружения в исходном тексте будет такой:

```
\begin{list}{заголовок_по_умолчанию}{команды}
....................................
\end{list}
```

Аргументы окружения `list` означают следующее. «Заголовок по умолчанию» — это заголовок элемента перечня, печатающийся в том случае, когда этот элемент перечня вводится командой `\item` без аргумента. Пример:

И какой-то малыш показал ему шиш.

И какой-то барбос укусил его в нос. Нехороший барбос, невоспитанный!

```
\begin{list}{И какой-то}{}
\item малыш показал ему шиш.
\item барбос укусил его в нос.
Нехороший барбос, невоспитанный!
\end{list}
```

Аргумент *команды* окружения `list` содержит те команды, которые будут исполнены после входа в перечень. Поэтому в нем можно задать команды, присваивающие новые значения параметрам оформления перечня (в частности, отбивкам, описанным в разд. 5.1: эти команды будут выполнены после команды `\@list...`, выполняющейся при входе в перечень). Кроме этого, во втором аргументе окружения `list` можно поместить команду `\usecounter`. Эта последняя требует одного обязательного аргумента — имени счетчика (счетчик должен быть определен). Если `\usecounter` присутствует во втором аргументе окружения `list`, то при входе в окружение значение счетчика, являющегося аргументом `\usecounter`, будет установлено в нуль, а каждая команда `\item` без аргумента будет увеличивать его на единицу с помощью `\refstepcounter` (так что на значения этого счетчика можно будет ссылаться с помощью `\label` и `\ref`). Вот пример с `\usecounter` (подразумевается, что у нас определен счетчик `tmp`):

Вот как выглядят первые буквы латинского алфавита:

A: Выглядит так же, как соответствующая русская буква, и читается так же.

B: Читается не так, как похожая на нее русская буква.

C: И с ней та же история.

```
Вот как выглядят первые буквы
латинского алфавита:
\begin{list}{\Alph{tmp}:}%
{\usecounter{tmp}}
\item Выглядит так же, как
соответствующая русская
буква, и читается так же.
\item Читается не так,
как похожая на нее
русская буква.
\item И с ней та же история.
\end{list}
```

Чтобы заголовки элементов перечня выравнивались по левому краю, а не по правому, можно завершить «заголовок по умолчанию» командой `\hfill`; чтобы по левому краю выравнивались заголовки, заданные в явном виде в необязательном аргументе команд `\item`, нужно завершить командой `\hfill` этот необязательный аргумент.

Окружением `list` разумно пользоваться не непосредственно, как в приведенных примерах, а для определения нового окружения с помощью `\newenvironment`. Вот, например, как в стандарте определяется окружение `quote`:

```
\newenvironment{\quote}%
{\begin{list}{}{\rightmargin=\leftmargin}\item[]}}%
{\end{list}}
```

Команда `\item` с пустым аргументом необходима, поскольку до команды `\item` в перечне не должно быть никакого текста (см. с. 132).

Наряду с окружением `list` в LaTeX'е определен его важный частный случай — окружение `trivlist`. Его отличия от `list` таковы:

- это окружение не требует аргументов (так же, как и все окружения для создания перечней, с которыми мы имели дело раньше);
- `\leftmargin`, `\labelwidth` и `\itemindent` для него всегда равны нулю (стало быть, текст печатается без втяжки); `\parsep` равно `\parskip`;
- команда `\item`, употребленная внутри этого окружения, обязана иметь аргумент (хотя бы пустой).

Как видите, многие возможности перечней в этом окружении не работают, но сохраняются такие важные черты, как `\topsep` (дополнительный интервал перед и после) плюс обычное свойство первой строки после перечня: она делается без абзацного отступа тогда и только тогда, когда после окружения в исходном тексте не оставлено пустой строки. Окружение `trivlist` также применяют обычно не само по себе, а для определения новых окружений; при этом в «открывающие команды» `\newenvironment` добавляют команду `\item[]`, а внутри окружения `\item` вообще не используют. Иногда используют и `\item` с аргументом (пример тому вы увидите ниже, в разд. 8.1).

Окружения `list` можно вкладывать друг в друга; максимальная глубина вложенности равна шести. В окружениях наподобие `itemize` предусмотрен «ограничитель», снижающий эту макимальную глубину до четырех. В личном стилевом файле, не рассчитанном на общее пользование, предусматривать такой ограничитель большого смысла нет: просто следите за тем, чтобы не вкладывать слишком много `list`'ов друг в друга.

6. Колонтитулы

Страницу документа, подготовленного с помощью LaTeX'а, можно рассматривать как состоящую из трех частей: тела страницы, верхнего колонтитула и нижнего колонтитула[4]. Мы знаем, что с помощью команды `\pagestyle` на оформление колонтитулов можно в какой-то мере влиять. Возможностей для этого, однако же, не слишком много. Сейчас мы увидим, как можно радикально менять вид колонтитулов.

За оформление верхних колонтитулов отвечают команды `\@oddhead` и `\@evenhead`. Точнее говоря, если стиль оформления документа «двусторонний», то команда `\@oddhead` задает верхний колонтитул на страницах с нечетными номерами, а команда `\@evenhead` — на страницах с четными номерами. Если же стиль оформления документа «односторонний», то `\@oddhead` задает все верхние колонтитулы, а команда `\@evenhead` на оформление документа вообще не влияет. Аналогично обстоит дело с `\@oddfoot` и `\@evenfoot`, отвечающими за нижние колонтитулы. Все четыре названные команды получают некоторое определение в LaTeX'овском стандарте, так что переопределять их надо с помощью `\renewcommand`.

Теперь обсудим, как вышеназванные команды влияют на оформление колонтитулов. Основной принцип таков. При оформлении страницы верхний колонтитул получается в результате исполнения команды

```
\hbox to\textwidth{\@evenhead}
```

(мы предположили, что стиль двусторонний и страница четная; в других случаях — с очевидными изменениями). Можно сказать, что в каждой из команд `\@evenhead` и ей подобных записан текст и TeX'овские команды, которые при верстке страницы будут подставлены в `\hbox to \textwidth`.

Пусть, например, мы готовим к изданию роман Л. Н. Толстого «Война и мир»; стиль документа выберем двусторонний. Предположим, что мы выбрали такое оформление:

- на левой странице разворота в верхнем колонтитуле написано имя автора, прижатое (выключенное) вправо;
- на правой странице разворота в верхнем колонтитуле написано название романа, прижатое (выключенное) влево;
- В нижних колонтитулах по центру расположен номер страницы, окруженный тире.

[4]Эта терминология не совпадает с традиционной, но удобна для наших целей.

Тогда надо переопределить команды для колонтитулов так:

```
\renewcommand{\@evenhead}{\hfil Л.\,Н.\,ТОЛСТОЙ}
\renewcommand{\@oddhead}{ВОЙНА И МИР\hfil}
\renewcommand{\@evenfoot}{\hfil --- \thepage ---\hfil}
\renewcommand{\@oddfoot}{\hfil --- \thepage ---\hfil}
```

Если мы хотим, чтобы каких-то колонтитулов вообще не было, то надо переопределить соответствующую команду на «ничего не делать», например, так:

```
\renewcommand{\@oddfoot}{}
```

Итак, мы научились менять оформление колонтитулов, переопределяя команду `\@evenhead` и ей подобные. А теперь — важное предупреждение: команда `\pagestyle` также переопределяет команды типа `\@evenhead`; если вы проведете переопределения в своем стилевом пакете, то первой же командой `\pagestyle` ваши переопределения будут отменены. Стало быть, если уж вы переопределяете команды наподобие `\@evenhead`, то после этого переопределения командой `\pagestyle` в документе пользоваться не надо (`\thispagestyle` можно).

Прежде чем перейти к более сложным вещам, скажем еще о трех стилевых параметрах. Во-первых, интервалы между колонтитулами и текстом регулируются параметрами со значением длины `\headsep`, задающим интервал между верхним колонтитулом и текстом, и `\footskip`, задающим расстояние между базисной линией последней строки в теле страницы и базисной линией нижнего колонтитула. Во-вторых, существует параметр `\headheight`, задающий высоту верхних колонтитулов. Если сумма высоты и глубины заданного вами колонтитула будет превышать `\headheight`, то при трансляции вы получите сообщение

```
Overfull \vbox occurred while \output was active.
```

Приведем пример, когда надо учитывать `\headheight`. Пусть мы хотим в том же издании «Войны и мира» отделять верхние колонтитулы от текста линейками. Разумный способ это сделать — передать в `\hbox to \textwidth` уже готовый блок шириной `\textwidth`, содержащий как текст колонтитула, так и линейку. Так как линейку под текстом удобно проводить в вертикальном режиме с помощью команды `\hrule`, в голову приходит вот что (здесь и далее мы для краткости приводим только определение `\@evenhead`):

```
\renewcommand{\@evenhead}%
{\vbox{\hbox to\textwidth{\hfil Л.\,Н.\,ТОЛСТОЙ}\hrule}}
```

У такого определения есть, однако, два недостатка. Во-первых, так как линейки в вертикальном режиме добавляются «впритык» к предшествующему блоку, у нас нет гарантии, что линейка не подойдет к тексту слишком близко; если бы к тому же текст в колонтитулах на разных страницах варьировался, как в дальнейших примерах, то может случиться и так, что две соседние линейки на развороте окажутся на разной высоте (из-за того, что в колонтитуле на одной странице будет буквы вроде у, опускающиеся ниже базисной линии, а на другой — нет). Средство от этого недостатка — добавить `\strut` в наш блок:

```
\renewcommand{\@evenhead}%
{\vbox{\hbox to\textwidth{\hfil \strut
Л.\,Н.\,ТОЛСТОЙ}\hrule}}
```

Второй недостаток — это то, что к размеру блока с текстом добавится ширина линейки, в результате чего этот размер превысит `\headheight` и мы будем получать уйму сообщений об overfull'е. Чтобы уйти от этого, надо еще чуть-чуть схитрить:

```
\renewcommand{\@evenhead}%
{\raisebox{0pt}[\headheight][0pt]{% начало блока
\vbox{\hbox to\textwidth{\hfil \strut
Л.\,Н.\,ТОЛСТОЙ}\hrule}}% конец блока
}% конец макроопределения
```

Понятно ли, что происходит? С помощью `\raisebox` мы заставляем TeX считать, что блок имеет высоту `\headheight` (нам даже незачем вникать, чему она фактически равна) и нулевую глубину (чтобы в сумме получилось то, что надо). Теперь ни о каких переполнениях речи не будет; пробел между колонтитулом и текстом можно при желании изменить, изменив значение `\headsep`.

Для нижних колонтитулов параметра, аналогичного `\headheight`, нет, так что смело делайте их любой высоты.

Кстати говоря, команду `\@evenhead` в этом примере можно было бы переопределить более простым образом, без `\vbox`, с использованием команды `\underline`. Наш способ, однако, более гибок: например, мы можем регулировать толщину линейки, чего с `\underline` не добьешься.

Итак, мы научились создавать собственные колонтитулы. Однако в LaTeX'овском стандарте в колонтитулы обычно помещается не только номер страницы, но и, например, номер и заглавие текущего раздела. Давайте научимся делать и это.

Чтобы передать в колонтитул какую-то информацию из текста, в LaTeX'е используются команды `\markboth` и `\markright`. Разберем, как они работают. Команда `\markboth`, требующая двух обязательных аргументов, запоминает пару «пометок» — два фрагмента текста (возможно, с TEX'овскими командами). Например, если мы скажем (где-нибудь между абзацами)

```
\markboth{Кот}{Пес}
```

то поместим между этими абзацами пару пометок: «левую пометку» `Кот` и «правую пометку» `Пес`. Сами по себе эти пометки никак не отражаются на печати. Однако же в определениях команд `\@oddhead` и ей подобных мы можем на эти пометки ссылаться. Именно, в этих определениях команда `\leftmark` дает левую пометку, а команда `\rightmark` — правую пометку. Например, если мы переопределим верхние колонтитулы как

```
\renewcommand{\@evenhead}{\leftmark\hfil}
\renewcommand{\@oddhead}{\hfil\rightmark}
```

то, начиная с той страницы, на которую попали наши пометки, в левом верхнем колонтитуле будет стоять выключенное влево слово «Кот», а в правом — выключенное вправо слово «Пес»[5]. Кстати, если никаких пометок в тексте нет, то как `\leftmark`, так и `\rightmark` дают «пустой» текст.

До сих пор мы молчаливо предполагали, что на каждой странице присутствует только одна пара пометок. Что будет, если таких пар пометок попадет на страницу несколько? Ответ: команда `\leftmark` в этом случае означает *левую* пометку из *самой верхней* пары пометок, попавших на страницу, а команда `\rightmark` означает *правую* пометку из *самой нижней* пары пометок, попавших на данную страницу. С другой стороны, если на страницу вообще ни одна пара пометок не попала, то `\leftmark` и `\rightmark` означают соответственно левую и правую пометки из последней пары пометок, встретившихся до этого.

Поясним сказанное примером. Пусть пометки

`\markboth{`x`}{`y`}` и `\markboth{`z`}{`t`}`

попали (в указанном порядке) на страницу 1, на страницу 2 вообще никаких пометок не попало, на страницу 3 попали целые три пары пометок:

`\markboth{`u`}{`v`}`, `\markboth{`a`}{`b`}` и `\markboth{`m`}{`n`}`,

[5]Точнее говоря, так будет, если в тексте нет команд типа `\section`: эти команды, как мы увидим ниже, автоматически вставляют в текст свои пометки, что усложняет картину.

а на страницах 4 и дальнейших никаких новых пометок не появлялось. Тогда значения команд `\leftmark` и `\rightmark` в процессе обработки этих страниц были таковы:

Страница	`\leftmark`	`\rightmark`
1	x	t
2	z	t
3	u	n
4 и далее	m	n

Наряду с командой `\markboth`, которая ставит в тексте новую пару пометок, есть еще и команда `\markright` с одним аргументом, которая ставит в тексте пару пометок таким образом: левый элемент в этой паре — такой же, как был в предыдущей паре пометок, а правый — тот, что задан в аргументе команды `\markright`. Например, если сначала идет команда

```
\markboth{Кот}{Пес}
```

а затем команда

```
\markright{Собака}
```

(и в промежутке между этими командами никаких других пометок в текст не вносится), то это равносильно тому, как если бы вторая из этих команд была

```
\markboth{Кот}{Собака}
```

Выше мы уже отмечали, что команды типа `\section` ставят в тексте пометки автоматически. Понять, как это делается и как можно влиять на этот процесс, проще всего на примере. Сделаем такие предположения о документе: класс — `article`, разделы и подразделы нумеруются (иными словами, значение счетчика `secnumdepth` больше единицы), действует стилевая опция `twoside` и в преамбуле была дана команда `\pagestyle{headings}`.

Во-первых, отметим, что в этом случае пометки в текст вставляют только команды `\section` и `\subsection`. При этом команда `\section` ставит пометки, автоматически выполняя команду `\sectionmark`, имеющую один обязательный аргумент — заглавие раздела (если команда `\section` была дана без необязательного аргумента) или вариант заглавия, заданный в необязательном аргументе команды `\section` (если таковой присутствует). Вот как определена `\sectionmark` (на место `#1` будет подставляться вариант заглавия, идущий в колонтитулы):

```
\newcommand{\sectionmark}[1]{\markboth{%
\MakeUppercase{\thesection\hspace{1em}#1}}% левая пометка
{}% правая пометка (она пуста)
}% конец макроопределения
```

Здесь используется LaTeX'овская команда `\MakeUppercase`, которую мы ранее не рассматривали. Не вдаваясь в подробности, скажем, что эта команда переводит все буквы в тексте[6], попавшем в ее аргумент, из строчных в прописные. Коль о том зашла речь, отметим, что есть еще и команда `\MakeLowercase`, которая наоборот переводит все буквы в тексте, попавшем в ее аргумент, из прописных в строчные. (Не пытайтесь, пожалуйста, использовать `\MakeUppercase` и `\MakeLowercase` вне аргументов `\markboth` или `\markright`, если не любите неприятных сюрпризов.) Если вы не хотите, чтобы в колонтитулах строчные буквы заменялись на прописные, можно при переопределении просто опустить команду `\MakeUppercase` (так и было сделано при оформлении книги, которую вы читаете).

Команда `\subsectionmark`, определяющая вид пометок, автоматически вставляемых в текст командой `\subsection`, определена следующим образом:

```
\newcommand{\subsectionmark}[1]{\markright
{\thesubsection\hspace{1em}#1}}
```

Здесь также аргумент `#1` — это заглавие подраздела (точнее, его вариант для колонтитулов).

Итак, в рассматриваемом нами случае команда `\section` вносит в текст следующую пару пометок: заглавие раздела, в котором все буквы заменены на прописные, в качестве левой пометки, и пустой текст в качестве правой пометки. Команда же `\subsection` вносит в текст пару пометок, в которой левая пометка такая же, как в предыдущей паре, а правая — заглавие подраздела (точнее, его вариант для колонтитулов), причем на сей раз «в натуральном виде», без замены строчных букв на прописные. Как уже отмечалось выше, команды `\subsubsection` и более мелкие никаких пометок в текст не вносят.

Посмотрим, как в этом стиле используются пометки. Колонтитулы на четных страницах в этом случае определены так:

```
\newcommand{\@evenhead}{\thepage\hfil
            \normalfont\slshape\leftmark}
```

Тем самым на страницах с четными номерами (они будут левыми на развороте) колонтитул будет выглядеть следующим образом: выключенный влево номер страницы (прямым шрифтом) и заглавие текущего

[6] Но не в именах команд; что будет с русскими буквами, зависит от русификации.

раздела (наклонным шрифтом, прописными буквами) — выключенное вправо[7]: ведь никаких других левых пометок в тексте нет!

Команда `\@oddhead`, отвечающая за верхние колонтитулы на страницах с нечетными номерами, определена в стиле `article` (при условии, что была команда `\pagestyle{headings}`) так:

```
\newcommand{\@oddhead}{{\normalfont\slshape\rightmark}%
             \hfil\thepage}
```

Стало быть, верхние колонтитулы к нечетным страницам выглядят так: название текущего подраздела наклонным шрифтом — выключенное влево, номер страницы прямым шрифтом — выключенный вправо. Впрочем, если в текущем разделе команд `\subsection` еще не было, то вместо заглавия будет пустое место, так как `\rightmark` будет пуста (определение команды `\sectionmark`, данное выше, показывает, что при исполнении команды `\section` в текст вносится пустая правая пометка, и пустой она остается до первой команды `\subsection`).

Приведем пример переопределения команд наподобие `\sectionmark`. Если нам не нравится, что в правом колонтитуле иногда бывает пустое место, то можно попросить команду `\section` вносить в текст непустую правую пометку — то же заглавие раздела. Для этого напишем в преамбуле так:

```
\newcommand{\sectionmark}[1]{\markboth
{\uppercase{\thesection\hspace{1em}#1}}% левая пометка
{\uppercase{\thesection\hspace{1em}#1}}% правая пометка
}% конец макроопределения
```

Теперь, если в разделе нет подразделов, то на правых страницах в верхнем колонтитуле будет также печататься заглавие текущего раздела. Оформление колонтитулов при этом будет стандартным. Если вы хотите еще и отойти от этого стандарта, то надо будет, вместо использования команды `\pagestyle`, напрямую переопределить команды типа `\@oddhead` и `\@evenhead`; надо думать, теперь вы найдете, как применить в этих определениях команды `\leftmark` и `\rightmark`.

Если в документе присутствует команда `\pagestyle` (с каким бы то ни было аргументом), то она отменит ваши переопределения команд типа `\sectionmark`. Поэтому все команды `\pagestyle` должны идти *до* ваших личных (пере)определений команд типа `\@evenhead`.

[7]Точнее говоря, это будет заглавие либо текущего раздела, либо первого из разделов, начинающихся на этой странице — см. выше обсуждение `\leftmark` и `\rightmark`.

Если вы вообще не хотите, чтобы при исполнении, скажем, команды `\subsection` в текст вносились какие-либо пометки, то можно «отключить» команду `\subsectionmark`, переопределив ее на «ничего не делать»:

```
\renewcommand{\subsectionmark}[1]{}
```

Бывает и так, что вас устраивает стандартный стиль оформления колонтитулов, но при этом не устраивает, что́ именно в эти колонтитулы автоматически записывается. Например, у ваших разделов длинные заглавия, и вы при этом хотите, чтобы они в несокращенном виде попали в оглавление, а сокращенные варианты заглавий пошли только в колонтитулы. Команда `\section` с необязательным аргументом тут не спасет, так как этот необязательный аргумент запишется и в оглавление тоже. Вот бы задать информацию для колонтитулов вручную, без того, чтобы это автоматически делали команды типа `\section`! Можно, конечно, переопределить на «ничего не делать» все команды типа `\sectionmark`, как в приведенном выше примере, но LaTeX предоставляет вам более простой способ. Если написать в стилевом файле (до того, как вы переопределяете команды наподобие `\@oddhead`)

```
\pagestyle{myheadings}
```

то все LaTeX'овские команды для создания разделов не будут вносить пометок в текст, а оформление колонтитулов будет стандартным. После этого можно самостоятельно переопределить `\@oddhead` и пр., если вам это нужно: пометки в текст командами рубрикации вноситься все равно не будут.

Теперь вы сможете, например, писать

```
\section{О некоторых специальных свойствах
подмножеств пустых множеств, не рассматривавшихся
в предыдущих разделах статьи}%
\markboth{\thesection\hspace{1em}Подмножества}{}
```

и в оглавлении будет заголовок целиком, а в колонтитуле — лишь слово «Подмножества» (мы подразумеваем, что стиль все тот же, только аргументом команды `\pagestyle` был `myheadings` вместо `headings`). Команда `\markboth` помещена *после* команды `\section`, чтобы номер раздела, генерируемый командой `\thesection`, был правильным. Кроме того, мы убрали с помощью знака процента пробел (конец строки) между командами `\section` и `\markboth`, чтобы пометки с гарантией попали на ту же страницу, что и заголовок раздела.

Другие команды, отвечающие за автоматическую расстановку пометок в тексте, устроены аналогичным образом: команда `\chapter` ставит пометки с помощью команды \chaptermark, команда \section — с помощью команды \sectionmark и т. д. — вообще, если команда, генерирующая раздел документа, определена с помощью \@startsection с первым аргументом abcd, то она будет ставить пометки с помощью команды \abcdmark. Все эти команды автоматически выполняются в процессе выполнения соответствующей команды, генерирующей раздел. Они должны иметь один обязательный аргумент, в качестве которого им передается заглавие раздела (точнее, вариант этого заглавия, предназначенный для колонтитулов и оглавления). Варианты «со звездочкой» команд, генерирующих разделы документа, никаких пометок в текст не вносят (как и следовало ожидать).

Если в аргументе команды \markboth или \markright присутствует не только текст, но и какие-то команды, то в пометки будут записаны не буквально эти команды, но их значение на момент запоминания пометок. Например, если в аргументе \markboth присутствует команда \thesection, то в пометку реально запишется (а потом и прочтется из \leftmark или \rightmark) номер раздела. Если же необходимо, чтобы какая-то команда записалась в пометку в том же виде, в каком она была задана в аргументе \markboth или \markright, то надо «защитить» ее, поставив перед ней \protect.

Еще один случай, когда пометки в текст вносятся LaTeX'ом автоматически, возникает при оформлении таких фрагментов документа, как оглавление, список иллюстраций или таблиц, список литературы и предметный указатель. Как именно они вносятся, зависит от класса документа и от того, пользовались ли мы (и как пользовались) командой \pagestyle. Именно, если класс — article, то никаких пометок при оформлении оглавления и т. п. в текст автоматически вноситься не будет; точно так же не будет этих пометок в любом классе после того, как выполнится команда \pagestyle с аргументом empty, plain или myheadings. С другой стороны, если класс документа — report или book, то, например, при исполнении команды \tableofcontents автоматически выполняется и команда

```
\markboth{\contentsname}
```

(с очевидными изменениями для списка иллюстраций и т. п.). Та же ситуация возникнет и после исполнения команды \pagestyle с аргументом headings (даже тогда, когда класс — article).

Итак, вы теперь имеете широкие возможности для создания собственного оформления колонтитулов. Во-первых, можно написать

```
\pagestyle{myheadings}
```

При этом внешний вид колонтитулов будет стандартный, но материал для колонтитулов вы будете поставлять вручную с помощью команд \markboth и/или \markright. Во-вторых, можно переопределять команды типа `\sectionmark`: при этом внешний вид колонтитулов будет по-прежнему стандартный, но материал для помещения в колонтитул будет отбираться автоматически по схеме, отличной от стандартной. В-третьих, можно переопределять команды типа `\@oddhead`: при этом вы меняете как стиль оформления колонтитулов, так и способ обращения с пометками, вносимыми в текст (в частности, если в ваших определениях команд типа `\@oddhead` не участвуют `\leftmark` и `\rightmark`, то эти пометки будут вообще проигнорированы). Наконец, можно объединить второй и третий подходы. При этом вы самостоятельно разрабатываете как внешний вид колонтитулов, так и то, какая информация из текста и как будет в них отражена.

7. Плавающие объекты

Этот раздел посвящен плавающим иллюстрациям и таблицам (короче: плавающим объектам).

7.1. Оформление подрисуночной подписи

Обсудим, как можно менять оформление подписи под рисунком (или таблицей), создаваемой командой `\caption`.

Над и под подписью предусмотрены вертикальные отбивки. Их размеры хранятся в параметрах со значением длины (растяжимой) `\abovecaptionskip` (отбивка над подписью) и `\belowcaptionskip` (под подписью). В стандарте отбивка над подписью равна `10pt` (без растяжимости или сжимаемости), а под подписью — нулю.

Чтобы изменить само оформление подписи, надо переопределить команду `\@makecaption`; вот ее стандартное определение в адаптированном виде (параметр `#1` — номер иллюстрации или таблицы с подписью, то есть значение команды `\thefigure` или `\thetable`; параметр `#2` — текст подписи):

```
\newcommand{\@makecaption}[2]{%
\vspace{\abovecaptionskip}%
\sbox{\@tempboxa}{#1: #2}
\ifdim \wd\@tempboxa >\hsize
   #1: #2\par
\else
   \global\@minipagefalse
   \hbox to \hsize {\hfil #1: #2\hfil}%
\fi
\vspace{\belowcaptionskip}}
```

Разберем этот код. Во второй и последней строчках вокруг подписи делаются отбивки, как было объяснено выше. В третьей строке в блоковую переменную `\@tempboxa` (эта переменная используется LaTeX'ом для временного хранения данных такого рода) записывается текст подписи вместе с номером (пока что в одну строку; поскольку в данный момент этот текст еще не печатается, нас не волнует, не окажется ли он длинней, чем ширина полосы). В следующих шести строках (с помошью не рассматривавшейся нами TeX'овской конструкции «условных макросов») длина подписи с номером сравнивается с шириной текста (она обозначена TeX'овским параметром `\hsize`: при одноколонном наборе это то же самое, что `\textwidth`, а при многоколонном — `\columnwidth`). Если подпись вместе с номером длиннее строки, то она печатается как абзац (часть кода между `\ifdim` и `\else`), а если не длиннее, то центрируется (часть кода между `\else` и `\fi`). Код на седьмой строке воспринимайте как данность.

Что можно изменить в этом определении? Поскольку `#1` — это номер, а `#2` — текст подписи, нетрудно заметить, что номер отделяется от подписи двоеточием, что с отечественными полиграфическими традициями не согласуется. Разумно заменить в этом месте двоеточие на точку. Кроме того, можно изменить шрифт, которым печатается подпись. Только не забудьте, что при переопределении надо будет вставить команды смены шрифта в три места (всюду, где в исходном определении стоит `#1: #2`): и в «измеряющую размер» строку, начинающуюся с `\sbox` (от смены шрифта размер тоже может измениться), и в оба варианта печати. Наконец, вы можете решить (не знаю, насколько это правильно) никогда не центрировать подписи; тогда в измененном определении `\@makecaption` надо будет удалить строки, начинающиеся с `\ifdim`, `\else` и `\fi`, а также текст между `\else` и `\fi` (и, возможно, установить свои любимые параметры верстки абзаца). Возможны, наверное, и другие решения. В любом случае помните, что текст между `\ifdim` и `\else` относится к случаю, когда подпись с номером в строку не умещается, а текст между `\else` и `\fi` — к противоположному.

И последнее: команду `\@makecaption` вы будете, конечно, переопределять с помощью `\renewcommand`; имейте в виду, что в данном случае надо пользоваться вариантом `\renewcommand` *без звездочки*, поскольку второй аргумент этой команды (текст подписи) вполне может состоять из нескольких абзацев.

7.2. Размещение плавающих объектов на странице

Сейчас мы обсудим параметры, которыми руководствуется LaTeX при размещении плавающих объектов на странице.

Сразу же отметим, что все эти параметры относятся в равной мере к плавающим иллюстрациям и плавающим таблицам, и любое изменение этих параметров также затрагивает плавающие объекты всех типов.

Часть параметров, отвечающих за плавающие объекты, представляет собой LaTeX'овские счетчики, которым можно присваивать новые значения командой `\setcounter`:

`topnumber` Максимальное количество плавающих объектов, которое разрешается разместить вверху страницы (при двухколонном наборе и иллюстрациях/таблицах шириной в колонку — вверху колонки). Значение по умолчанию: 2.

`bottomnumber` Максимальное количество плавающих объектов, которое разрешается разместить внизу страницы (при двухколонном наборе и иллюстрациях/таблицах шириной в колонку — внизу колонки). Значение по умолчанию: 1.

`totalnumber` Максимальное количество плавающих объектов, которое разрешается разместить на странице (при двухколонном наборе и иллюстрациях/таблицах шириной в колонку — в колонке). Значение по умолчанию: 3.

`dbltopnumber` При наборе в две колонки: максимальное количество плавающих объектов шириной во всю страницу, которое разрешается разместить вверху страницы. Значение по умолчанию: 2.

Значение счетчика `totalnumber` не влияет на количество иллюстраций и/или таблиц на странице, специально для них предназначенной (таковая, как мы помним, выделяется, если в необязательном аргументе окружения `figure` или `table` присутствует буква `p`).

Вторая группа параметров регулирует уже не количество плавающих иллюстраций на странице, а величину места, ими занимаемого. Все эти параметры являются командами наподобие `\arraystretch` или `\baselinestretch`; чтобы менять значения параметров, надо их переопределять с помощью команды `\renewcommand`. Эти параметры таковы:

`\topfraction` Максимальная доля страницы, которую могут занимать плавающие объекты, размещаемые вверху страницы. Значение по умолчанию: 0.7. Это означает, что плавающие иллюстрации и таблицы, размещаемые вверху страницы, могут занимать не более 70% страницы по высоте. Если мы хотим уменьшить эту долю, скажем, до 50%, надо написать

```
\renewcommand{\topfraction}{0.5}
```

Ниже слова «доля страницы» также всюду означают «доля страницы по высоте».

`\bottomfraction` Максимальная доля страницы, которую могут занимать плавающие объекты внизу страницы. Значение по умолчанию: 0.3.

`\textfraction` *Минимальная* доля страницы, которую должен занимать текст, а не плавающие объекты (к страницам, создаваемым LaTeX'ом специально для размещения плавающих объектов при обработке необязательного аргумента `p`, это не относится). Значение по умолчанию: 0.2.

`\floatpagefraction` Этот параметр, напротив, относится именно к страницам, которые LaTeX создает при обработке необязательного аргумента `p`. Он указывает минимально возможную долю такой страницы, которую могут занимать размещаемые на ней плавающие иллюстрации и таблицы. Значение по умолчанию: 0.5 (стало быть, специальная страница для плавающих иллюстраций, которую вы требуете с помощью необязательного аргумента `p` к окружению `figure`, не будет создана, пока эти иллюстрации занимают менее 50% высоты страницы текста).

Все вышеуказанные параметры применимы также к двухколонному набору и плавающим объектам шириной в колонку. Если же набор — в две колонки, а ширина плавающего объекта — целая страница, то используются такие параметры:

`\dbltopfraction` То же, что `\topfraction`, применительно к иллюстрациям (таблицам) шириной в целую страницу при двухколонном наборе. Значение по умолчанию: 0.7.

`\dblfloatpagefraction` То же, что `\floatpagefraction`, применительно к иллюстрациям (таблицам) шириной в целую страницу при двухколонном наборе. Значение по умолчанию: 0.5.

При вычислении, какую часть страницы занимает плавающий объект, учитывается не только размер собственно иллюстрации или таблицы, но и величина вертикальных отступов («отбивок»), отделяющих этот объект от остального текста. Эти отступы задаются следующей группой параметров. Все эти параметры суть длины, присваивать им новое значение надо с помощью `\setlength`.

`\textfloatsep` Отступ между текстом и иллюстрациями (таблицами), размещаемыми вверху или внизу страницы.

`\floatsep` Отступ между двумя иллюстрациями (таблицами).

`\intextsep` Отступ между текстом и иллюстрациями (таблицами), размещаемыми посреди страницы (при обработке необязательного аргумента `h`).

Как водится, три перечисленных параметра применимы и к иллюстрациям (таблицам) шириной в колонку при двухколонном наборе. А вот как обозначаются соответствующие параметры, если набор в две колонки, а иллюстрация или таблица занимает по ширине всю страницу:

`\dbltextfloatsep` При двухколонном наборе — отступ между текстом и иллюстрацией (таблицей), занимающей всю страницу по ширине.

`\dblfloatsep` При двухколонном наборе — отступ между двумя иллюстрациями (таблицами), занимающими всю страницу по ширине.

Если вы почему-либо решите менять эти параметры с помощью команды `\setlength`, полезно иметь в виду, что при пользовании шрифтом кегля 11 значения `\floatsep`, `\intextsep` и `\dblfloatsep` равны `12pt plus 2pt minus 2pt`, а значения `\textfloatsep` и `\dbltextfloatsep` равны `20pt plus 2pt minus 4pt`.

По умолчанию плавающие объекты не отделены от текста ничем, кроме вышеперечисленных отбивок. Но можно сделать и так, чтобы иллюстрации (таблицы) отделялись от текста как-то иначе (линейками, например). Для этих целей предусмотрены следующие три команды:

`\topfigrule` Разделитель между плавающим объектом, размещаемым вверху страницы (колонки), и остальным текстом.

`\botfigrule` Разделитель между текстом и размещаемым внизу страницы (колонки) плавающим объектом.

Эти две команды относятся и к двухколонному набору, при условии, что иллюстрации (таблицы) занимают по ширине одну колонку (иными словами, если используются окружения `figure` или `table` без звездочек). А если используются варианты этих окружений со звездочками, то используется еще один параметр:

`\dblfigrule` То же, что `\topfigrule`, для случая, когда набор двухколонный, а плавающий объект занимает по ширине всю страницу.

По умолчанию указанные три команды ничего не делают. Если вы хотите задать явные разделители между текстом и иллюстрациями (таблицами), то эти команды надо определить с помощью `\newcommand`.[8]

[8]Именно так, а не переопределить с помощью `\renewcommand`!

Определять эти команды нужно не произвольным образом: чтобы они правильно стыковались с LaTeX'овскими алгоритмами размещения плавающих объектов, нужно иметь в виду следующее:

1) каждая из этих команда будет выполняться в те моменты, когда TeX находится в вертикальном режиме;

2) по окончании работы каждой из этих команд TeX должен снова оказаться в вертикальном режиме;

3) текст, генерируемый каждой из этих команд, не должен, с точки зрения TeX'а, занимать места по вертикали.

Поэтому будет неправильно, если вы, решив отделять текст от иллюстраций линейкой, определите `\botfigrule` просто как `\hrule`: первое и второе условия при этом выполнены будут, а вот третье — нет, в результате чего LaTeX собьется со счета при решении вопроса о размещении иллюстраций. Формально правильным было бы такое решение:

```
\newcommand{\botfigrule}{\hrule\vspace{-0.4pt}}
```

(вспомним, что линейка, генерируемая командой `\hrule`, имеет по умолчанию толщину `0.4pt`). Впрочем, формальной правильности мало: если вы опробуете такое определение на практике, то увидите, что линейка вплотную прилегает к иллюстрации, что никуда не годится. Правильно действовать, например, так:

```
\newcommand{\botfigrule}{\vspace{-3pt}\hrule
    \vspace{2.6pt}}
```

Теперь мы проводим линейку не прямо по верхней кромке иллюстрации, а на три пункта выше; заключительное `\vspace{2.6pt}` нужно для того, чтобы в сумме получилось нулевое вертикальное смещение.

С `\topfigrule` и `\dblfigrule` вы сможете теперь разобраться самостоятельно.

В заключение отметим, что разделители, определяемые `\topfigrule` и ей подобными командами, не обязаны быть именно линейками: необходимо только при их определении учитывать три перечисленных выше обстоятельства.

8. Разное

8.1. Теоремы, выключные формулы

Чтобы изменить оформление «теорем» (окружений, определяемых с помощью `\newtheorem`), надо переопределить команды `\@begintheorem` и `\@opargbegintheorem` (первая из них отвечает за оформление «теорем» без необязательного аргумента, вторая — за оформление «теорем» с необязательным аргументом). Первая из них определена так:

```
\newcommand{\@begintheorem}[2]{\begin{trivlist}
\item[\hspace{\labelsep}{\bfseries#1\ #2}]
\itshape}
```

Здесь аргумент `#1` означает название «теоремы» (например, «Теорема», «Предложение», «Лемма»,... — в команде `\newtheorem` это слово являлось вторым обязательным аргументом), а аргумент `#2` означает номер «теоремы». Если мы, например, хотим, чтобы после номера «теорем» стояла точка, нам достаточно переопределить эту команду, добавив точку после `#2`. Что делать для того, чтобы сменить шрифт, которым печатаются номер или текст «теорем», также достаточно ясно.

Команда `\@opargbegintheorem` определяется так:

```
\newcommand{\@opargbegintheorem}[3]{\begin{trivlist}
\item[\hspace{\labelsep}{\bfseries #1\ #2\ (#3)}]
\itshape}
```

Здесь `#1` и `#2` по-прежнему означают название и номер «теоремы», а `#3` — необязательный аргумент «теоремы» (обычно в качестве такового задается имя ученого, которому приписывается данная теорема).

Если оформление, задаваемое окружением `trivlist`, вас не устраивает, то можно переопределить две вышеуказанные команды более радикально. Общий принцип таков. Перед текстом «теоремы», не имеющей необязательного аргумента, исполняется команда `\@begintheorem`; у этой команды должно быть два аргумента, причем первый из них — название «теоремы», а второй — ее номер. Если «теорема» имеет необязательный аргумент, то вместо `\@begintheorem` перед ее текстом исполняется команда `\@opargbegintheorem`, имеющая три аргумента: первые два — такие же, как у `\@begintheorem`, и третий — необязательный аргумент «теоремы» (имя первооткрывателя). Наконец, после текста «теоремы» исполняется команда `\@endtheorem`, которая изначально определена очень просто:

```
\newcommand{\@endtheorem}{\end{trivlist}}
```

В принципе можно переопределить все три эти команды, чтобы получить свое оформление «теорем» (например, в духе наших макросов для автоматической нумерации задач из гл. VII). Только следите, чтобы переопределения всех трех команд были согласованы друг с другом: если, например, вы изгоните `\begin{trivlist}` из определения `\@begintheorem`, но при этом оставите команду `\@endtheorem` в неприкосновенности, то на каждой «теореме» LaTeX будет сообщать вам об ошибке (отсутствие баланса команд `\begin` и `\end`).

При пользовании AMS'овскими классами документов (приложение Г) «теоремы» определяются иначе. Вряд ли, впрочем, вы сочтете нужным переделывать стандарт Американского математического общества.

Теперь скажем кое-что про стиль оформления номеров выключных формул, заданных в виде окружения `equation`. Как вам уже известно, можно переопределить команду `\theequation` или (с помощью команды `\@addtoreset`) изменить подчиненность счетчика `equation`; при этом изменится оформление самих номеров формул. Кроме этого, можно изменить то, что печатается возле этих номеров. Для этого следует переопределить команду `\@eqnnum`. Изначально она определена так:

```
\newcommand{\@eqnnum}{(\theequation)}
```

При желании можно заменить тут круглые скобки на что-нибудь другое. Имейте в виду, что номер выключной формулы обрабатывается TeX'ом «в математическом режиме», как формула (латинские буквы по умолчанию набираются «математическим курсивом»и т. п.).

8.2. Сноски

Стиль оформления сносок зависит от многих вещей. Начнем с пробела между страницей и сносками. Чтобы его изменить, надо применить команду `\setlength` с необычным первым аргументом. Вот, например, как выглядит команда, устанавливающая стандартную для LaTeX'а величину этого пробела:

```
\setlength{\skip\footins}{12pt plus 4pt minus 4pt}
```

Почему в первом аргументе`\setlength` целых две команды и что они означают, объяснить в рамках этой книги невозможно, так что воспринимайте этот рецепт для установки пробела между текстом и сносками догматически (для любознательных: вся правда содержится в пятнадцатой главе книги [2]).

Далее, сноски обычно отделяются от текста не только пробелом, но и линейкой. Чтобы задать вид этой линейки, отличный от стандартного, либо задать какой-то другой разделитель, надо переопределить команду `\footnoterule`, которая в стандарте определена так:

```
\newcommand{\footnoterule}{\vspace*{-3pt}
     \hrule width .4\columnwidth
     \vspace*{2.6pt}}
```

К этому макроопределению необходим комментарий: непонятно, зачем нужны команды `\vspace`. Дело в том, что «текст», генерируемый командой `\footnoterule`, не должен, с точки зрения TeX'a, занимать места по вертикали (фактически он располагается внутри пробела между текстом и сносками, о котором шла речь выше). Поэтому мы сначала отступаем на 3 пункта вверх, затем печатаем линейку (вспомним, что по умолчанию линейка, генерируемая командой `\hrule`, имеет толщину 0.4 пункта), а затем спускаемся на 2.6 пункта вниз. В итоге получается, что и линейка напечаталась, и места по вертикали мы не занимаем, поскольку $-3 + 0.4 + 2.6 = 0$.

Может возникнуть вопрос, зачем нужен `\vspace*{-3pt}`: не проще ли обойтись без этой команды, а после `\hrule` сказать `\vspace*{-0.4pt}`? Ответ: в этом случае линейка напечаталась бы вплотную к сноске. Ср. с. 332.

Если вы хотите изменить ширину или толщину линейки, команду `\footnoterule` можно переопределить; только не забудьте проследить, чтобы отрицательный `\vspace` скомпенсировал толщину линейки. Можно, собственно говоря, сделать так, чтобы этой линейки вообще не было, сказав

```
\renewcommand{\footnoterule}{}
```

(уж тут-то места по вертикали мы не займем!). Если вам вдруг понадобится задать совсем иной разделитель между сносками и текстом, можете переопределить команду `\footnoterule` принципиально по-иному. В этом случае необходимо знать следующее:

1) команда `\footnoterule` будет выполняться в те моменты, когда TeX находится в вертикальном режиме;

2) по окончании работы команды `\footnoterule` TeX должен снова оказаться в вертикальном режиме;

3) текст, генерируемый командой `\footnoterule`, не должен, с точки зрения TeX'a, занимать места по вертикали.

Следующий параметр, от которого зависит оформление сносок, — это параметр со значением длины `\footnotesep`. Он означает следующее. В начале каждой сноски, для того чтобы линейка, отделяющая сноски от текста, не подходила к тексту слишком близко, вставляется невидимая линейка нулевой ширины наподобие `\strut` (см. разд. III.10.3). Так вот, `\footnotesep` задает высоту этой линейки.

За вид номеров сносок в тексте отвечает команда `\@makefnmark`. По умолчанию она определена следующим образом:

```
\newcommand{\@makefnmark}{\hbox{\mathsurround=0pt
$^{\@thefnmark}$}}
```

Здесь на место команды `\@thefnmark` при выполнении будет подставлен номер сноски (или то, что его заменяет, если мы пользовались командой `\footnotemark`). Обратите внимание, что номер сноски оформлен как верхний индекс в математической формуле — именно благодаря этому номера сносок печатаются над строкой. По этой же причине внутри группы, являющей собой аргумент команды `\hbox`, устанавливается в нуль параметр `\mathsurround` — иначе, если вы установили для него ненулевое значение, номер сноски будет окружен лишними пробелами.

И, наконец, самое главное — команда, генерирующая собственно текст сноски. Она называется `\@makefntext`. Вот ее стандартное определение, в котором аргумент `#1` обозначает текст сноски, а команда `\@thefnmark` означает то же, что и выше:

```
\newcommand{\@makefntext}[1]{\parindent=1em\noindent
          \hbox to 1.8em{\hss\@makefnmark}#1}
```

При переопределении этой команды следует иметь в виду, что она будет выполняться внутри аргумента команды `\parbox` с длиной строки, равной ширине колонки текста; в приведенном выше определении применена команда `\noindent`, чтобы подавить абзацный отступ в первом абзаце сноски, в котором будет печататься ее номер.

Имейте в виду, что поскольку текст сноски, являющийся аргументом команды `\@makefntext`, может состоять из нескольких абзацев, переопределять эту команду надо с помощью `\renewcommand` без звездочки.

8.3. Список литературы

Если вам не нравится, что библиографические ссылки печатаются в квадратных скобках, то вы можете переопределить команды `\@cite` и `\@biblabel`. По умолчанию они определены так:

```
\newcommand{\@cite}[2]{[{#1\if@tempswa , #2\fi}]}
\newcommand{\@biblabel}[1]{[#1]\hfill}
```

Мы не будем даже пытаться объяснить, что́ означают заковыристые команды в первом из этих макроопределений (`#1` в нем означает номер источника, а `#2` — дополнительную информацию, помещенную в необязательном аргументе команды `\cite`, если таковой имеется), а отметим только, что в «замещающем тексте» первого из этих определений можно заменить квадратные скобки на какие-нибудь другие (скажем, круглые или косые) — и тогда в соответствующих скобках будет печататься ссылка, генерируемая командой `\cite`; аналогично, если заменить скобки в «замещающем тексте» второго из этих определений, то в соответствующих скобках будет печататься номер источника в списке литературы. Разумеется, при переопределении этих команд надо писать `\renewcommand` вместо `\newcommand`.

Если вы хотите, чтобы дополнительная информация печаталась вне скобок, скажите

```
\renewcommand{\@cite}[2]{{[#1]\if@tempswa , #2\fi}}
```

(не делайте пробела между закрывающей скобкой и `\if@tempswa`, иначе на печати выйдет лишний пробел). Текст между `\if@tempswa` и `\fi` будет выполняться тогда и только тогда, когда у команды `\cite` присутствует необязательный аргумент; если в вашем определении `\@cite` не будет участвовать `#2`, то необязательный аргумент команды `\cite` будет игнорироваться.

8.4. Предметный указатель

Первое, что вы можете изменить в оформлении предметного указателя (окружение `theindex`), — это отступы, создаваемые командами `\item`, `\subitem` и т. д. Для изменения отступов «на первом уровне» (создаваемых командой `\item`) надо переопределить команду `\@idxitem` (но *не* сам `\item`!). Чтобы вам было от чего отталкиваться, посмотрите на стандартное определение этой команды. Оно очень простое:

```
\newcommand{\@idxitem}{\par\hangindent=40pt}
```

Для изменения отступов, создаваемых такими командами, как `\subitem` и `\subsubitem`, надо переопределить непосредственно эти команды. Их стандартные определения также бесхитростны:

```
\newcommand{\subitem}{\par\hangindent=40pt\hspace*{20pt}}
\newcommand{\subsubitem}{\par\hangindent=40pt\hspace*{30pt}}
```

Наконец, команда `\indexspace`, создающая в предметном указателе дополнительный вертикальный пробел, определяется в стандартных стилях так:

```
\newcommand{\indexspace}{\par\vspace{10pt plus 5pt minus 3pt}}
```

Иногда хочется изменить оформление предметного указателя более существенным образом. Например, вы можете захотеть, чтобы ссылка на предметный указатель присутствовала в оглавлении (в LaTeX'овском стандарте это не предусмотрено); возможно также, что вы захотите предпослать предметному указателю небольшое введение, набранное во всю ширину страницы. Чтобы добиться таких вещей, надо переопределить окружение `\theindex`. Его стандартное определение в стилях `book` и `report` выглядит так:

```
\newenvironment{theindex}{\@restonecoltrue
\if@twocolumn\@restonecolfalse\fi
\columnseprule=0pt \columnsep=35pt
\twocolumn[\@makeschapterhead{\indexname}]%
\@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
\thispagestyle{plain}\parindent=0pt
\setlength{\parskip}{0pt plus .3pt}%
\let\item=\@idxitem}%
{\if@restonecol \onecolumn \else \clearpage \fi}
```

Первая, вторая, предпоследняя и последняя строки этого определения содержат незнакомые вам команды; мы не будем пытаться объяснить, что они значат, а только скажем, что менять эти места в определении окружения `theindex` нельзя. Зато все остальное в этом определении должно быть понятно читателю, усвоившему основную часть нашей книги. Именно, в третьей строке задаются параметры двухколонного оформления в предметном указателе: там сказано, что колонки в окружении `theindex` не надо разделять линейкой и задается промежуток между двумя колонками (см. разд. IV.4.1). В пятой строке дана команда, задающая (неким не рассматривавшимся нами способом) материал для колонтитулов. Она либо вносит левую и правую пометки, совпадающие со стандартным заглавием предметного указателя, либо ничего не делает. Если вы переопределяете `\theindex`, то можете в этой строке написать `\markboth` вместо `\@mkboth` с теми аргументами, с какими считаете нужным, или вообще убрать эту строку, чтоб пометок не было. В шестой строке обратите внимание на команду, устанавливающую нулевое значение абзацного отступа. Менять ее не надо, поскольку именно с таким значением абзацного отступа согласовано действие команд `\item`, `\subitem` и т. п. (вспомните, как действует команда `\hangindent`).

Наконец, в четвертой строке стоит команда `\twocolumn` с необязательным аргументом. Как объяснялось в разд. III.9.6, то, что стоит в необязательном аргументе, будет напечатано во всю ширину страницы. В стандартном определении тут стоит команда `\@makeschapterhead`, создающая заголовок ненумерованной главы (см. разд. 3), но вы можете записать туда и любой текст, который хотите предпослать собственно предметному указателю. Правда, тут возникает один технический момент: введение к предметному указателю вы, видимо, захотите редактировать, и нехорошо для этого всякий раз лазать в стилевой пакет. Один из возможных выходов таков. Запишите в вашем определении окружения `theindex` четвертую строку так:

```
\twocolumn[\@makeschapterhead{\indexname}\input{ukaz.tex}]
```

Здесь `ukaz.tex` — файл, в который вы запишете свое введение к предметному указателю.

Если вы хотите, чтобы предметный указатель был отражен в оглавлении, то в необязательный аргумент команды `\twocolumn` надо добавить команду `\addcontentsline`, например, в таком виде:

```
\addcontentsline{toc}{chapter}{\indexname}
```

Поскольку предметный указатель, задаваемый с помощью окружения `theindex`, использует команду `\twocolumn`, колонки на последней странице указателя могут оказаться разной высоты (см. с. 149). Как обычно, лекарство от этого — подключить стилевой пакет `multicol` и сделать так, чтобы окружение `theindex` использовало для печати окружение `multicols`. Для этого нужно переопределить окружение `theindex` следующим образом (мы предполагаем, что пакет `multicol` подключен):

```
\renewenvironment*{theindex}{\columnseprule=0pt\columnsep=35pt
\@makeschapterhead{\indexname}%
\@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
\thispagestyle{plain}\parindent=0pt
\setlength{\parskip}{0pt plus .3pt}%
\let\item=\@idxitem
\begin{multicols}{2}}%
{\end{multicols}}
```

Разумеется, вы можете также изменить параметры в этом определении по своему усмотрению.

Последнее, что нужно сказать про окружение `theindex`, это то, что в классе `article` четвертая строка его стандартного определения выглядит так:

```
\twocolumn[\section*{\indexname}]%
```

(поскольку в классе `article` главы не определены).

Приложение A

Архитектура TeX'а и LaTeX'а

В этом приложении не содержится никаких практических рецептов. Его цель в том, чтобы читатель, прочитавший основную часть книги, получил общее представление о структуре работающего комплекта TeX'а.

1. Немного истории

Как гласит легенда, Дональд Кнут (Donald E. Knuth) написал программу TeX для себя, чтобы готовить к печати книги серии «Искусство программирования для ЭВМ». Первая версия TeX'а появилась в 1978 году; начиная с 1989 года все изменения в TeX'е сводятся в основном к исправлению ошибок (Кнут сознательно не стремится совершенствовать TeX дальше, чтобы предотвратить распространение несовместимых версий). Тому, кто первым обнаружит очередную ошибку, Кнут выплачивает денежную премию; время от времени ставка удваивается, и тем не менее Кнут отнюдь не разорился[1].

Программа TeX распространяется свободно, а ее исходные тексты полностью доступны: помимо книги [2], обязательной для изучения каждому, кто собирается стать TeXником, в виде книги опубликован и полный текст программы с комментариями. Текст программы написан на изобретенном Кнутом псевдокоде под названием Web; раньше Web транслировали в Pascal (с помощью программы, написанной все тем же Кнутом), в настоящее время используется транслятор из Web'а в язык C.

[1]На данный момент сумма составляет \$ 327.68, или 2^{15} центов; исходная сумма равнялась \$ 2.56.

2. Макропакеты и форматы

Если получить программу TeX из кнутовского исходного текста (то есть перевести псевдокод в Pascal или C, а затем скомпилировать), то использовать полученную программу («чистый TeX» — по-английски говорят еще «virgin TeX») для обработки текстов будет еще невозможно: чистый TeX не знает смысла ни спецзнаков[2], ни макросов. В него встроены только «примитивные команды» (с. 240).

Стало быть, прежде чем использовать TeX для дела, его надо обучить смыслу тех многочисленных макросов, которые используются в реальных `tex`-файлах. Набор TeX'овских макросов, используемый для обработки текстов, называется *макропакетом*. С одним из TeX'овских макропакетов вы уже неплохо знакомы: это тот самый LaTeX, которому посвящена вся эта книга.

Как же TeX узнает определения макросов из используемого вами макропакета? Ответ таков. Среди известных чистому TeX'у примитивных команд есть и команда, предназначенная для определения макросов (чем-то похожая на известные вам `\newcommand` и `\renewcommand`); если перед началом файла с текстом и формулами, который вы хотите обработать, приписать «исходный текст макропакета» — список всех используемых в макропакете (скажем, том же LaTeX'е) макроопределений — и передать полученный файл для обработки программе TeX, то TeX сначала узнает смысл используемых вами макросов, а к моменту начала собственно текста будет уже «во всеоружии» и благополучно создаст нужный вам `dvi`-файл.

На практике, однако, так никогда не делается: макропакеты имеют достаточно большой размер, и обработка их TeX'ом даже на современных компьютерах происходит не мгновенно, а уж в конце семидесятых годов потери времени при работе по такой схеме были бы совершенно неприемлемы. Поэтому Кнут применил в этом месте следующую оптимизацию: при установке TeX'а для работы с каким-то макропакетом TeX считывает исходный текст этого макропакета и записывает содержимое своих внутренних таблиц в специальный файл, называемый *форматным*; при запуске для работы с текстом TeX предварительно считывает информацию о макросах, содержащуюся в макропакете, из форматного файла (или попросту «формата», как иногда говорят), что существенно быстрее, чем всякий раз обрабатывать исходный текст макропакета заново.

(Строго говоря, сказанное в предыдущем абзаце не вполне соответствует действительности. Использование файлов формата — не только оптимизация, поскольку таблицы переносов, необходимые для верстки

[2] Кроме символа «\».

абзацев, могут быть обработаны TEX'ом только на стадии генерации форматного файла.)

Первый макропакет написал одновременно с программой TEX сам Кнут: это пакет Plain TEX, который также описан в книге [2].

Следующим после Plain TEX'а макропакетом, получившим распространение, стал $\mathcal{AMS}$-TEX, созданный Майклом Спиваком (Michael Spivak). По сравнению с Plain TEX'ом в него были добавлены удобные средства для набора сложных формул, особенно многострочных. Подчеркнем, что всех $\mathcal{AMS}$-TEX'овских эффектов можно добиться и в Plain TEX'е — возможности TEX'а новый макропакет расширить не может, — но там это требует громоздких записей и довольно серьезного знания TEX'овских внутренних механизмов; Спивак создал для этих целей удобные сокращенные обозначения (то есть макроопределения!), которыми математики стали с удовольствием пользоваться (и до сих пор пользуются).

В 1984 году Лесли Лэмпорт (Leslie Lamport) создал макропакет LATEX (его заключительная версия, вышедшая в 1989 году, называется LATEX 2.09). Возможно, одной из важнейших новых черт LATEX'а явилась возможность автоматической нумерации и — главное — автоматической генерации ссылок с помощью команд `\label`, `\ref` и `\pageref`. Достижение этого эффекта средствами Plain TEX'а требует весьма изощренного программирования.

Наконец, в 1995 году появилась новая версия LATEX'а, а именно тот самый LATEX 2_ε, описанию которого посвящена вся эта книга. Одним из первоначальных толчков к коренной переработке LATEX'а было желание включить в него возможности $\mathcal{AMS}$-TEX'а, в частности, возможность пользоваться символьными шрифтами Американского математического общества (в «старом» LATEX'е добавлять новые шрифты сверх стандартного комплекта было довольно неудобно). В настоящее время поддержкой и совершенствованием системы LATEX 2_ε занимается группа TEXнологов (Frank Mittelbach, Michel Goossens и другие); после первоначального периода быстрых изменений LATEX 2_ε практически стабилизировался, хотя раз в год и выходит новая версия.

Подведем предварительные итоги: при практической работе программу TEX запускают не просто так, но с указанием форматного файла, соответствующего используемому макропакету; форматные файлы генерируются при установке TEX'а из исходных текстов макропакетов. Отметим еще, что исходные тексты макропакетов переносимы между платформами (DOS, Linux и т. п.), а форматные файлы зависят от реализации TEX'а.

3. Реализации TEX'а

Как мы уже говорили, программа TEX (то есть интерпретатор языка TEX) написана Кнутом не на каком-то из использовавшихся в то время языков программирования, но на специальном псевдокоде. Благодаря этому TEX можно сравнительно легко приспособить к различным типам компьютеров и операционных систем, и эти реализации будут работать одинаково. (Чтобы достичь единообразия, Кнут практически нигде не пользуется вычислениями с плавающей запятой: на разных компьютерах они могут дать разные результаты, а это, в свою очередь, может повлиять на верстку.)

Конечно, пользовательский интерфейс и интерфейс с операционной системой в разных реализациях устроены по-разному, но при обработке одного и того же `tex`-файла с одним и тем же макропакетом на любой платформе получится один и тот же `dvi`-файл, при печати которого получится один и тот же текст.

Реализации TEX'а имеют свои названия. Например, долгое время была популярна созданная Эберхардом Маттесом (Eberhard Mattes) реализация TEX'а для DOS, называемая emTEX. Современная реализация TEX'а для UNIX (в частности, Linux, FreeBSD и др.) называется teTEX. Для Windows имеются реализации fpTEX и MiKTEX. Наконец, для Макинтошей существуют по крайней мере две реализации TEX'а: коммерческая, называемая TEXtures, и shareware-вариант OzTEX.

Не следует путать реализации с макропакетами: в принципе в любой из реализаций TEX'а можно установить любой макропакет (в частности, Plain TEX, LATEX и AMS-TEX установлены всюду). Поэтому на задаваемый иногда новичками вопрос: «А у тебя LATEX или MiKTEX?» ответить невозможно.

4. Шрифты и `dvi`-драйверы

Как мы уже говорили, программа TEX (с тем или иным макропакетом) читает TEX-файл и преобразует его в файл с расширением `.dvi` (от слов «device independent»). Этот файл содержит информацию о том, какие буквы из каких шрифтов и в каком месте страницы надо разместить. (Любопытные могут изучить структуру этого файла подробнее; для начала посмотрите, что выдает программа `dvitype`, которая переводит информацию из `dvi`-файла в текстовую форму.)

Заметим, что в `dvi`-файлах указаны лишь номера букв, но не сказано, что, собственно говоря, надо напечатать на бумаге или показать на экране. На самом деле TEX этого и не знает: с его точки зрения каждая буква — это просто прямоугольник.

Откуда же берутся шрифты, используемые TEX'ом? Первоначальный комплект шрифтов создал сам Дональд Кнут. Для этого он вместе со своими коллегами

- разработал специальный формат, в котором записываются данные о размерах букв (`tfm`);
- разработал специальный формат, в котором записываются данные о форме букв (`pk`, есть также промежуточный формат `gf`); эти данные для каждой буквы указывают матрицу из черных и белых точек, которая должна быть напечатана на месте этой буквы;
- разработал специальный язык (METAFONT), на котором удобно описывать буквы, и программу METAFONT (сокращенно `mf`), которая по такому описанию изготавливает `tfm`-файлы, а также `pk`-файлы для разных разрешений (измеряемых обычно числом точек на дюйм);
- написал описание семейства шрифтов Computer Modern на языке METAFONT(все шрифты, использованные до сих пор в этой книге, в своей латинской части повторяют шрифты Computer Modern);
- подготовил к изданию три книги, содержащие подробное описание программы METAFONT, ее комментированный текст, а также тексты шрифтов семейства Computer Modern (в дополнение к двум другим книгам: описанию TEX'а и комментированному тексту программы TEX!).

Вот пример: описание шрифта Computer Modern Roman (прямой светлый, как в основном тексте книги) размера 10 pt на языке METAFONT содержится в файле `cmr10.mf` (точнее, этот файл устанавливает значения параметров и затем отсылает к другим файлам). Программа METAFONT, запущенная на этом файле с указанием разрешения 600 dpi (оно предусмотрено во многих современных принтерах), порождает файл `cmr10.tfm` (который на самом деле не зависит от разрешения) и файл `cmr10.600pk`.

Из этих двух файлов TEX использует лишь первый. Второй понадобится, когда `dvi`-файл, ссылающийся на шрифт cmr10, будет фактически печататься на принтере (с разрешением 600 dpi). Для принтера с другим разрешением может потребоваться вновь запустить программу METAFONT, указав тот же файл `cmr10.mf`, но другое разрешение.

Помимо шрифта cmr10, в стандартный комплект TEXа входят шрифты других размеров (cmr5, cmr6, cmr7, cmr8, cmr9, cmr12, cmr17). Кроме того, можно масштабировать шрифты (при этом TEX использует тот же самый `tfm`-файл, но другой `pk`-файл, потому что число точек в увеличенном шрифте другое).

Вернемся к `dvi`-файлам. Как мы уже говорили, в результате обработки TEX'ом `tex`-файла получается `dvi`-файл. Формат его стандартизирован и не зависит ни от использованного макропакета, ни от реализации TEX'а. Этот файл содержит указания о размещении символов на странице. В `dvi`-файл записываются только названия используемых шрифтов, но не их начертания.

В свою очередь, `dvi`-файл обрабатывается с помощью программы, называемой `dvi`-драйвером, которая осуществляет печать, показ текста на экране и т. п. (для разных устройств и разных нужд есть разные драйверы); `dvi`-драйвер в процессе работы использует растровые описания шрифтов, то есть `pk`-файлы.

Например, драйвер `dvihplj`, входящий в состав системы emTEX (для DOS), преобразует `dvi`-файл в файл в языке PCL (который понимают многие лазерные принтеры). Драйвер `xdvi`, входящий в состав системы teTEX (для UNIX), показывает `dvi`-файл на экране, используя вызовы оконной системы X Window System (в этом контексте слово «Window» не имеет никакого отношения к системе с похожим названием одной известной фирмы). И так далее.

Современные `dvi`-драйверы действуют обычно по следующей схеме: если готового `pk`-файла нет, они запускают программу METAFONT, которая из `mf`-файла генерирует этот `pk`-файл с нужным разрешением.

Некоторые реализации TEX'а идут в этом отношении еще дальше: если в процессе обработки `tex`-файла встречается запрос на шрифт, для которого нет соответствующего `tfm`-файла, то вызывается METAFONT, который недостающий `tfm`-файл и генерирует (по канону в такой ситуации TEX должен выдавать сообщение об ошибке).

Строго говоря, TEX может работать с любыми шрифтами, надо только, чтобы для них были соответствующие `tfm`-файлы и чтобы соответствующие имена шрифтов были предусмотрены в используемом макропакете. В частности, довольно часто с TEX'ом используют так называемые «Type 1 PostScript-шрифты». О них, а также о языке PostScript, более подробно говорится в приложении Б.

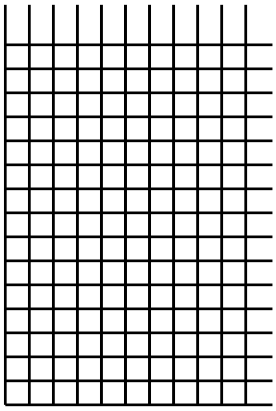
Приложение Б

PostScript и TeX

1. Что такое PostScript?

Язык программирования PostScript разработан фирмой Adobe прежде всего для программирования принтеров. Вот пример файла на языке PostScript и порождаемой им картинки:

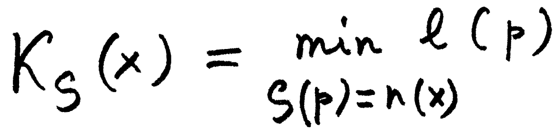
```
%!PS
%%BoundingBox: 0 0 110 150
0 9 9 10 mul {
  dup 0 moveto
  150 lineto
} for stroke
0 9 9 15 mul {
  dup 0 exch moveto
  100 exch lineto
} for stroke
```



Если вы когда-либо имели дело со стековыми языками программирования (скажем, с языком FORTH), то догадаетесь, что команда `dup` удваивает вершину стека, а команда `exch` меняет местами два верхних элемента. Если же нет, тоже ничего страшного — эти знания нам не пригодятся. Важно лишь понимать, что программа на языке PostScript представляет собой файл, который при исполнении порождает рисунок (из одной или нескольких страниц). Этот рисунок вовсе не обязан состоять из линий. Например, PostScript-файл

```
%!PS-Adobe-2.0 EPSF-2.0
%%Creator: pnmtops
%%Title: kolm.ps
%%Pages: 1
%%BoundingBox: 197 353 415 408
%%EndComments
/readstring {
  currentfile exch readhexstring pop
} bind def
/picstr 227 string def
%%EndProlog
%%Page: 1 1
gsave
197.28 353.64 translate
217.44 54.72 scale
1812 456 1
[ 1812 0 0 -456 0 456 ]
{ picstr readstring }
image
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
        <...>
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffe00
0000010000001fffffffc0007fffffffe000000000000bfffffff80001ff
ffffffffffff837fffffffffffffe0000fffffc000fffffffffc001fffff
ffe0000000007ffffff80001ffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffff0ffff
        <...>
ffffffffffffffffffffffff
grestore
showpage
%%Trailer
```

(мы опустили большую часть строк — всего в файле их более трех тысяч) порождает рисунок



(определение колмогоровской сложности из рукописи Андрея Николаевича Колмогорова 1970 года; файл был получен ее сканированием).

Структура этого файла понятна: строки из букв кодируют черные и белые точки (и буквы `f` составляют большинство: точки в основном белые).

Аналогично можно задавать и полутоновые рисунки, только нужен очень качественный принтер (или еще более дорогое фотонаборное устройство), чтобы они хорошо печатались.

В язык PostScript включен некоторый минимальный набор шрифтов. Вот пример файла, ссылающегося на встроенные шрифты, и соответствующего рисунка:

```
%!PS
%%BoundingBox: 0 0 160 40
/Times-Roman findfont 24
scalefont setfont
5 5 moveto
(This is a string) show
```

This is a string

К сожалению, русских букв в шрифтах стандарт языка PostScript не предусматривает, так что если заменить `This is a string` на что-нибудь русское, ничего хорошего не получится.

Да и вообще разных шрифтов предусмотрено немного. Зато стандарт PostScript предусматривает возможность определять свои шрифты, и в этих шрифтах уже могут быть любые буквы, в том числе русские. Например, нехитрая картинка

Это строка (string)!

может быть задана таким файлом:

```
%!PS-Adobe-2.0 EPSF-2.0
%%Creator: dvips(k) 5.86 Copyright 1999 Radical Eye Software
%%Title: exampl4.dvi
%%BoundingBox: 148 654 241 666
%%EndComments
%DVIPSWebPage: (www.radicaleye.com)
%DVIPSCommandLine: dvips -E -o exampl4.ps exampl4.dvi
%DVIPSParameters: dpi=600, compressed
%DVIPSSource:  TeX output 2002.07.21:2027
%%BeginProcSet: texc.pro
%!
/TeXDict 300 dict def TeXDict begin/N{def}def/B{bind def}N/S{exch}N/X{S
N}B/A{dup}B/TR{translate}N/isls false N/vsize 11 72 mul N/hsize 8.5 72
mul N/landplus90{false}def/@rigin{isls{[0 landplus90{1 -1}{-1 1}ifelse 0
0 0]concat}if 72 Resolution div 72 VResolution div neg scale isls{
landplus90{VResolution 72 div vsize mul 0 exch}{Resolution -72 div hsize
mul 0}ifelse TR}if Resolution VResolution vsize -72 div 1 add mul TR[
matrix currentmatrix{A A round sub abs 0.00001 lt{round}if}forall round
exch round exch]setmatrix}N/@landscape{/isls true N}B/@manualfeed{
statusdict/manualfeed true put}B/@copies{/#copies X}B/FMat[1 0 0 -1 0 0]
```

```
N/FBB[0 0 0 0]N/nn 0 N/IEn 0 N/ctr 0 N/df-tail{/nn 8 dict N nn begin
/FontType 3 N/FontMatrix fntrx N/FontBBox FBB N string/base X array
/BitMaps X/BuildChar{CharBuilder}N/Encoding IEn N end A{/foo setfont}2
array copy cvx N load 0 nn put/ctr 0 N[}B/sf 0 N/df{/sf 1 N/fntrx FMat N
df-tail}B/dfs{div/sf X/fntrx[sf 0 0 sf neg 0 0]N df-tail}B/E{pop nn A
definefont setfont}B/Cw{Cd A length 5 sub get}B/Ch{Cd A length 4 sub get
}B/Cx{128 Cd A length 3 sub get sub}B/Cy{Cd A length 2 sub get 127 sub}
B/Cdx{Cd A length 1 sub get}B/Ci{Cd A type/stringtype ne{ctr get/ctr ctr
1 add N}if}B/id 0 N/rw 0 N/rc 0 N/gp 0 N/cp 0 N/G 0 N/CharBuilder{save 3
1 roll S A/base get 2 index get S/BitMaps get S get/Cd X pop/ctr 0 N Cdx
0 Cx Cy Ch sub Cx Cw add Cy setcachedevice Cw Ch true[1 0 0 -1 -.1 Cx
sub Cy .1 sub]/id Ci N/rw Cw 7 add 8 idiv string N/rc 0 N/gp 0 N/cp 0 N{
rc 0 ne{rc 1 sub/rc X rw}{G}ifelse}imagemask restore}B/G{{id gp get/gp
gp 1 add N A 18 mod S 18 idiv pl S get exec}loop}B/adv{cp add/cp X}B
/chg{rw cp id gp 4 index getinterval putinterval A gp add/gp X adv}B/nd{
/cp 0 N rw exit}B/lsh{rw cp 2 copy get A 0 eq{pop 1}{A 255 eq{pop 254}{
A A add 255 and S 1 and or}ifelse}ifelse put 1 adv}B/rsh{rw cp 2 copy
get A 0 eq{pop 128}{A 255 eq{pop 127}{A 2 idiv S 128 and or}ifelse}
ifelse put 1 adv}B/clr{rw cp 2 index string putinterval adv}B/set{rw cp
fillstr 0 4 index getinterval putinterval adv}B/fillstr 18 string 0 1 17
{2 copy 255 put pop}for N/pl[{adv 1 chg}{adv 1 chg nd}{1 add chg}{1 add
chg nd}{adv lsh}{adv lsh nd}{adv rsh}{adv rsh nd}{1 add adv}{/rc X nd}{
1 add set}{1 add clr}{adv 2 chg}{adv 2 chg nd}{pop nd}]A{bind pop}
forall N/D{/cc X A type/stringtype ne{]}if nn/base get cc ctr put nn
/BitMaps get S ctr S sf 1 ne{A A length 1 sub A 2 index S get sf div put
}if put/ctr ctr 1 add N}B/I{cc 1 add D}B/bop{userdict/bop-hook known{
bop-hook}if/SI save N @rigin 0 0 moveto/V matrix currentmatrix A 1 get A
mul exch 0 get A mul add .99 lt{/QV}{/RV}ifelse load def pop pop}N/eop{
SI restore userdict/eop-hook known{eop-hook}if showpage}N/@start{
userdict/start-hook known{start-hook}if pop/VResolution X/Resolution X
1000 div/DVImag X/IEn 256 array N 2 string 0 1 255{IEn S A 360 add 36 4
index cvrs cvn put}for pop 65781.76 div/vsize X 65781.76 div/hsize X}N
/p{show}N/RMat[1 0 0 -1 0 0]N/BDot 260 string N/Rx 0 N/Ry 0 N/V{}B/RV/v{
/Ry X/Rx X V}B statusdict begin/product where{pop false[(Display)(NeXT)
(LaserWriter 16/600)]{A length product length le{A length product exch 0
exch getinterval eq{pop true exit}if}{pop}ifelse}forall}{false}ifelse
end{{gsave TR -.1 .1 TR 1 1 scale Rx Ry false RMat{BDot}imagemask
grestore}}{{gsave TR -.1 .1 TR Rx Ry scale 1 1 false RMat{BDot}
imagemask grestore}}ifelse B/QV{gsave newpath transform round exch round
exch itransform moveto Rx 0 rlineto 0 Ry neg rlineto Rx neg 0 rlineto
fill grestore}B/a{moveto}B/delta 0 N/tail{A/delta X 0 rmoveto}B/M{S p
delta add tail}B/b{S p tail}B/c{-4 M}B/d{-3 M}B/e{-2 M}B/f{-1 M}B/g{0 M}
B/h{1 M}B/i{2 M}B/j{3 M}B/k{4 M}B/w{0 rmoveto}B/l{p -4 w}B/m{p -3 w}B/n{
p -2 w}B/o{p -1 w}B/q{p 1 w}B/r{p 2 w}B/s{p 3 w}B/t{p 4 w}B/x{0 S
rmoveto}B/y{3 2 roll p a}B/bos{/SS save N}B/eos{SS restore}B end

%%EndProcSet
TeXDict begin 40258437 52099154 1000 600 600 (exampl4.dvi)
@start
%DVIPSBitmapFont: Fa zcr10 10 16
/Fa 16 253 df<121C127FEAFF80A8EA7F00AB123EAB121CABC7FCA8121C127FEAFF80A5
EA7F00121C093C79BB17>33 D<146014E0EB01C0EB0380EB0700130E131E5B5BA25B485A
A2485AA212075B120F90C7FCA25A121EA2123EA35AA65AB2127CA67EA3121EA2121F7EA2
7F12077F1203A26C7EA26C7E1378A27F7F130E7FEB0380EB01C0EB00E01460135278BD20
>40 D<12C07E12707E7E7E120F6C7E6C7EA26C7E6C7EA21378A2137C133C133E131EA213
1F7FA21480A3EB07C0A6EB03E0B2EB07C0A6EB0F80A31400A25B131EA2133E133C137C13
78A25BA2485A485AA2485A48C7FC120E5A5A5A5A5A13527CBD20>I<ED03F090390FF00F
F890393FFC3C3C9039F81F707C3901F00FE03903E007C03A07C003E010000FECF000A248
486C7EA86C6C485AA200075C6C6C485A6D485A6D48C7FC38073FFC38060FF0000EC9FCA4
120FA213C06CB512C015F86C14FE6CECFF804815C03A0F80007FE048C7EA0FF0003E1403
```

```
48140116F8481400A56C1401007C15F06CEC03E0003F1407D80F80EB0F80D807E0EB3F00
3901FC01FC39007FFFF0010790C7FC26387EA52A>103 D<EA0380EA0FE0487EA56C5AEA
0380C8FCAAEA03F012FFA312071203B3AA487EB512C0A312387EB717>105
D<3903F00FF000FFEB3FFCECF03F9039F1C01F803A0FF3800FC03803F70013FE496D7EA2
5BA35BB3A3486C497EB500C1B51280A329257EA42E>110 D<3807E01F00FFEB7FC09038
E1E3E09038E387F0380FE707EA03E613EE9038EC03E09038FC0080491300A45BB3A2487E
B512F0A31C257EA421>114 D<EBFF03000313E7380F80FF381E003F487F487F00707F12
F0A2807EA27EB490C7FCEA7FE013FF6C13E06C13F86C7F00037FC67F01071380EB007F14
1F00C0EB0FC01407A26C1303A37E15806C13077EEC0F00B4131E38F3C07C38E1FFF038C0
3F801A277DA521>I<1318A51338A31378A313F8120112031207001FB5FCB6FCA2D801F8
C7FCB215C0A93800FC011580EB7C03017E13006D5AEB0FFEEB01F81A347FB220>I<EB1F
E0EBFFFC3803E03F3907000F80390F8007E0486C6C7E13E06E7EA26E7E6C5A6C5AC8FCA4
147FEB07FFEB3FE0EBFE00EA03F8EA0FF0EA1FC0123F485A90C7FC160C12FEA31401A26C
13036CEB077C903980063E18383FC01E3A0FE0781FF03A03FFF00FE03A007F8007C02627
7DA52A>193 D<B539C001FF80A32607F800EBF8006C48EB03E0ED0780030EC7FC5D5D15
F0EC01C04A5A4AC8FC140E141E147F5C01F17F9038F39FC09038F70FE013FE496C7E9038
F003F86E7EA26E7E157F6F7E151F826F7E82486CEB1FFCB539C03FFFC0A32A247EA32E>
203 D<EB03FE90380FFF8090383E03E09038F800F84848137C48487F48487F4848EB0F80
001F15C090C712074815E0A2007EEC03F0A400FE15F8A9007E15F0A2007F14076C15E0A2
6C6CEB0FC0000F15806D131F6C6CEB3F006C6C137EC66C13F890387E03F090381FFFC0D9
03FEC7FC25277EA52A>207 D<3903F01FE000FFEB7FF89038F1E07E9039F3801F803A0F
F7000FC0D803FEEB07E049EB03F04914F849130116FC150016FEA3167FAA16FEA3ED01FC
A26DEB03F816F06D13076DEB0FE001F614C09039F7803F009038F1E07E9038F0FFF8EC1F
C091C8FCAB487EB512C0A328357EA42E>210 D<EB03FC90381FFF8090387E03E03901F8
0070484813F83907E001FC380FC003A2EA1F80123F90380001F848EB00F01500A2127E12
FEAA127E127FA26C14067F001F140E6D130C000F141C6C6C13386C6C13706C6C13E03900
7C07C090381FFF00EB07F81F277DA525>I<007FB612FCA3397E007E000078153C007015
1C0060150CA200E0150EA2481506A5C71400B114FF90B6FCA327247DA32E>I<0018EB1F
F8D81C01B5FCD81E0714E03A1F1F801FF89039FC0003FC01F0EB00FF01C06E7E496E7E90
C8EA0FE0001E82707E001C6F7E707E1218707EA2CAEA7F80A218C0173F18E0A3171F18F0
A591B7FCA391C8121FA518E0173FA318C0001E167FD87F801680487EEFFF00A24C5AA24C
5A494A5A007EC85B00704B5A6C4B5A6CED7F80000F4BC7FCD807E0EB03FCD801FEEB1FF8
D8007FB512E0010F91C8FC9038007FF8343D7EBA3C>252 D E
%EndDVIPSBitmapFont
end
%%EndProlog
%%BeginSetup
%%Feature: *Resolution 600dpi
TeXDict begin

%%EndSetup
1 0 bop 639 523 a Fa(\374\324)q(\317)28 b(\323\324\322)r(\317\313\301)f
(\(string\)!)p eop
%%Trailer
end
userdict /end-hook known{end-hook}if
%%EOF
```

Видно, что этот файл изготовлен уже не вручную (как первый и третий примеры этого приложения), а с помощью программ.

Для начала мы применили LaTeX к файлу `exampl4.tex` такого содержания:

```
\documentclass{article}
\pagestyle{empty}
\begin{document}
Это строка (string)!
\end{document}
```

Получился файл exampl4.dvi, который в свою очередь был обработан командой

```
dvips -E -o exampl4.ps exampl4.dvi
```

Программа dvips (о которой мы еще будем говорить) и изготовила нам файл exampl4.ps (приведенный выше). Мы не беремся объяснить смысл входящих в этот файл команд, но видно, что вначале идут комментарии (строки с процентами), затем определения вспомогательных команд (занимающие целую страницу), затем (после пустой строки) идет описание шрифта zcr10 (основной русский шрифт в использованной нами русификации), и лишь потом идут собственно изображаемые символы (последние строки файла). При этом латинские буквы слова string так прямо и записаны в файле, а русские буквы заданы своими кодами. Например, \317 (что означает восьмеричный код 317) соответствует букве «о» и встречается дважды: как третья буква в слове «Это» и как четвертая буква в слове «строка». (Знатоки смогут заключить отсюда, что в используемых нами шрифтах скорее всего применяется так называемая «кодировка koi8-r».)

Таким образом, на PostScript-принтере можно напечатать текст любым шрифтом, только надо предварить этот текст описанием шрифта.

Вероятно, вы уже догадались, что примеры программ на языке PostScript приведены лишь для удовлетворения любопытства читателя; при практической работе PostScript-файлы создаются и обрабатываются программами; читать их невооруженным глазом (и писать их вручную) вам, скорее всего, никогда не придется. Но некоторое представление о возможностях языка PostScript иметь всё же полезно.

2. Драйвер dvips

В предыдущем разделе мы уже упоминали программу dvips. Эта программа получает на вход dvi-файл и преобразует его в PostScript-файл. При этом она использует pk-файлы для шрифтов, ссылки на которые есть в dvi-файле, и включает описания этих шрифтов (или только нужных букв из этих шрифтов) в PostScript-файл. Таким образом, имея dvi-файл, программу dvips и PostScript-принтер, можно получить печатный текст: надо лишь[1] направить результат работы программы dvips в принтер.

При этом изготовление PostScript-файла и его печать могут быть разделены во времени и пространстве. Можно положить PostScript-файл статьи в архив препринтов, чтобы и через много лет (какой автор не мечтает об этом?) интересующиеся читатели его списывали и читали. Можно изготовить файл в одной стране и послать его по электронной почте в издательство другой страны.

Заметим, что для этого не нужно, чтобы в издательстве умели пользоваться системой TEX — вполне достаточно, чтобы там умели печатать PostScript-файлы.

Приведем примеры использования программы dvips. Команда

```
dvips -o article.ps article.dvi
```

преобразует файл article.dvi в PostScript-файл article.ps. Команда

```
dvips -o article.ps -p 5 -l 7 article.dvi
```

помещает в PostScript-файл только три страницы (с пятой по седьмую); указав только -p 5, можно напечатать все страницы с пятой до конца, а указав только -l 7, можно напечатать все страницы с начала до седьмой включительно. Команда

```
dvips -D 300 -o article.ps article.dvi
```

указывает, что нужно использовать (и включить в PostScript-файл) шрифты с разрешением 300 dpi (по умолчанию обычно используются шрифты 600 dpi); это можно быть полезно для старых принтеров. Ключ -E мы уже видели выше, он применим к файлам из одной страницы и обрезает их по краям (в качестве BoundingBox берется минимальный блок, содержащий текст). Наконец, иногда полезен ключ -e 0, который говорит, что при вычислении положений букв не надо полагаться на округления, произведенные в pk-шрифте (это полезно, если без этого получаются нерегулярные промежутки между буквами).

3. Ghostscript

Но как напечатать PostScript-файл, если принтер не понимает языка PostScript? (Как правило, дешевые лазерные принтеры, а также пода-

[1]Это не всегда просто. В системах типа DOS или Windows иногда может помочь команда copy /b <filename> prn; для использования PostScript-файлов в этих системах при отсутствии PostScript-принтера полезны программы Ghostscript и GSView, см. приложение Ж.

вляющее большинство струйных и матричных принтеров его не понимают). Кроме того, неэкономно (и неэкологично) каждый раз печатать текст, когда хочется проверить, как он будет выглядеть после очередных изменений.

Фирма Adobe, которая разработала язык PostScript, написала интерпретатор этого языка, который можно встраивать в другие продукты (принтеры, программы). Например, существует программа Adobe Acrobat, которая умеет показывать на экране PostScript-файлы и печатать их на разных принтерах (то есть преобразовывать в другие форматы, доступные этим принтерам).

К сожалению, продукты фирмы Adobe дороги, часто требуют для работы (также дорогой) системы Windows, а воровать (как в теории признавал, кажется, даже Остап Бендер) грешно. К счастью, разработчики свободно распространяемого программного обеспечения о нас позаботились. Peter Deutsch и его коллеги написали свободно распространяемую программу `Ghostscript`, которая представляет собой интерпретатор языка PostScript, умеющий работать с самыми разными принтерами (лазерными, струйными, матричными), а также выводить файлы в разнообразных форматах (включая PDF — Portable Document Format, также разработанный фирмой Adobe). При этом используются любезно предоставленные фирмой URW шрифты. Эта программа приспособлена практически для всех операционных систем (Linux, FreeBSD, другие виды UNIX, DOS, Windows и др.) и стала стандартом de facto. (Например, организаторы некоторых конференций просят, чтобы присылаемые на рецензию статьи в формате PostScript правильно обрабатывались этой программой.)

Программа `Ghostscript` является составной частью программ, показывающих PostScript-файлы на экране (популярные программы такого типа — `gv`, `ghostview`, `GSView`). Обычно в такие программы встроена также возможность печати всего текста (текущей страницы, выбранного множества страниц) на принтере. Именно эти программы вы скорее всего будете использовать на практике; их интерфейс зависит от операционной системы, и мы скажем про него в приложении Ж.

4. PostScript-рисунки

Если вы используете программу `dvips`, появляется возможность соединять набранный в программе TEX текст с PostScript-рисунками. (Именно таким способом было подготовлено это приложение.) Вот как это делается.

Прежде всего, надо подключить стилевой пакет `graphicx` с опцией `dvips`. Последняя буква в названии пакета именно `x`; существует вариант этого пакета, называющийся `graphics`, но он немного отличается от описываемого нами. Для подключения пакета `graphicx` с опцией `dvips` следует написать в преамбуле

```
\usepackage[dvips]{graphicx}
```

Затем надо подготовить собственно рисунок в виде PostScript-файла. Точнее говоря, требуется специальный тип PostScript-файлов, который называется Encapsulated PostScript (традиционное расширение: `.eps`). Такие файлы предназначены для включения в другие файлы, и потому подчиняются дополнительным ограничениям (например, там не должно быть нескольких страниц).

Как проще всего изготовить Encapsulated PostScript-файл, зависит от типа рисунка и ваших навыков. Например, автограф Колмогорова был просканирован (в системе Linux) с помощью программы `xsane`, затем полученный `pgm`-файл (полутоновая матрица) был почищен и обрезан по краям в программе `gimp`, затем преобразован в черно-белый (без полутонов) программой `pgmtopbm`, затем преобразован в PostScript программой `pnmtops`, и в результате получился файл `exampl2.ps`.[2] В тексте мы написали

```
\begin{center}
\includegraphics{exampl2.ps}
\end{center}
```

Здесь `\includegraphics` — команда (определенная в пакете `graphicx`), а ее аргумент — имя файла с рисунком. Команда `\includegraphics` имеет множество необязательных параметров. Например, картинку с текстом «Это строка (string)!» мы включили с увеличением в два раза, сказав

```
\begin{center}
\includegraphics[scale=2]{exampl4.ps}
\end{center}
```

Можно также повернуть рисунок (скажем, на 60°), написав

```
\begin{center}
\includegraphics[angle=60]{exampl4.ps}
\end{center}
```

[2]Если честно, то после всего этого в последних строках файла `exampl2.ps` все буквы были заменены на `f` с помощью текстового редактора, чтобы убрать грязь в низу рисунка.

С точки зрения TEX'а фрагмент `\includegraphics[...]{...}` ведет себя как блок. Его размеры берутся из самогó включаемого файла (из строки BoundingBox, где единицей измерения является 1/72 дюйма, а четыре числа означают левую, нижнюю, правую и верхнюю координаты; шириной блока считается разность между правой и левой, а высотой — между верхней и нижней координатами). В принципе можно вручную такую строку добавить (если почему-либо изготовленный вами PostScript-файл ее не содержит) или исправить (если указанные там числа вас не устраивают).

Но того же самого эффекта можно добиться и средствами TEX'а, используя команды `\vbox` и `\hbox`. Вот, например, как была помещена рисованная буква «В» в начало одного из разделов:

```
\newsavebox{\bukva}
\sbox{\bukva}{\includegraphics{litera.ps}}
\begin{wrapfigure}{l}{\wd\bukva}
\vbox to 13mm{\vss\usebox{\bukva}\vspace*{-7mm}}
\end{wrapfigure}
предыдущем разделе мы уже упоминали
программу \texttt{dvips}...
```

Горизонтальные размеры буквы мы не меняли, а вертикальный размер и положение регулировали (`13mm` и `-7mm` были подобраны; команда `\vss` означает бесконечно сжимаемый и растяжимый клей, используемый в вертикальном режиме). Сам файл был получен сканированием иллюстрации из книги А. Г. Шицгала «Русский типографский шрифт»; эта иллюстрация, в свою очередь, представляет собой воспроизведение фрагмента из «Остромирова Евангелия» (рукописной книги XI века). Подрисуночная подпись (которую можно было бы сделать обычным способом, с помощью `\caption`), естественно, в этом случае была бы неуместна.

Если у документа (или у пакета `graphicx`) указана опция `draft` (иными словами, слово `draft` присутствует среди необязательных аргументов; по поводу опции `draft` см. с. 158), то вместо PostScript-рисунков, включенных с помощью `\includegraphics`, будет печататься прямоугольник того же размера с названием PostScript-файла. (Это может сэкономить время и краситель в принтере.)

Сказанного достаточно, чтобы включать рисунки в статьи и книги. Но всё же полезно понимать более конкретно, что происходит в этом месте в `dvi`-файле. Он *не* включает в себя описание рисунка. Зато он включает специальное указание (конструкция `special`), которое программа `dvips` интерпретирует как необходимость включить в это место текста фрагмент из PostScript-файла, указанного рядом со special. Тем самым программа dvips нуждается не только в dvi-файле, но и в PostScript-файлах рисунков. (Кстати, они нужны и LATEX'у, хотя из них он читает лишь строку BoundingBox, чтобы определить размер блока.) Зато готовый PostScript-файл (выдаваемый программой dvips) уже самодостаточен; все необходимые рисунки включены в него и больше никаких файлов не надо.

Подчеркнем еще раз обстоятельство, которое внимательный читатель уже понял: использование PostScript-рисунков выходит за рамки официального стандарта TEX'а, поэтому могут существовать (и существуют) dvi-драйверы, которые попросту игнорируют соответствующие конструкции special. При их использовании рисунков не будет видно. К счастью, программа dvips стала одним из самых популярных dvi-драйверов, и в расчете на нее можно смело использовать пакет graphicx.

В заключение вернемся к исходному вопросу: как же включить рисунок в текст? Мы знаем, как включить PostScript-файл, но как получить этот файл? Если рисунок сканируется, то такой файл можно получить с помощью программ обработки изображений. Мы уже кратко упомянули о них, говоря об автографе Колмогорова; все упомянутые программы есть в системах типа UNIX, но и в Microsoft Windows для этого есть средства, в частности, входящие в комплект MiKTEX свободно распространяемые программы, см. раздел Ж.2.

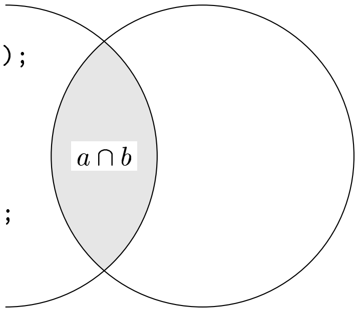
Важно иметь в виду, что рисунок в формате .eps имеет некоторый размер в абсолютных единицах длины (в его заголовке указаны координаты Bounding Box, при этом единица принята равной 1/72 дюйма). С другой стороны, рисунки в растровых форматах имеют обычно размеры, измеряемые в точках. Поэтому при их преобразовании в .eps есть произвол: размер точки может быть взят любым. При печати .eps-файла (в составе текста) на принтере происходит обратное преобразование, поскольку принтер в реальности печатает черные и белые точки, и итоговое преобразование сводится к масштабированию растрового рисунка (с неизбежной при этом интерполяцией, если коэффициент масштабирования не равен 1). Заметим, что коэффициент масштабирования пропорционален значению параметра scale в команде \includegraphics.

Кроме того, если исходный рисунок был полутоновой (о цветных мы не говорим, там всё еще сложнее), то в какой-то момент он должен быть тем или иным способом преобразован в черные и белые точки (большинство принтеров умеет печатать только их). Это может делаться и в принтере, и до него (в интерпретаторе Ghostscript); кроме того, можно не доверять этим механизмам и пытаться с самого начала преобразовать полутоновой рисунок в двуцветный, например, с помощью программы `pgmtopbm`. Какой из этих многочисленных вариантов даст при печати лучшие результаты, зависит от конкретной ситуации. (Еще сложнее выбрать наилучший способ, если готовится оригинал-макет для типографии.)

Все сказанное относилось к сканированным оригиналам рисунков. Но при изготовлении чертежей и технических рисунков мы не советуем чертить их на бумаге и потом сканировать. Есть гораздо более удобные средства. Одно из них (кстати, входящее в большинство TEX-комплектов) — программа `metapost` (или `mpost`, иногда `mp`). Ее автор — John Hobby — переделал программу METAFONT так, чтобы вместо шрифтов она порождала PostScript-файлы с рисунками. Эта программа заслуживает отдельной небольшой книжки (которой на русском языке пока нет; есть авторское английское описание, которое обычно входит в комплект TEX'а в виде файла `mpman.ps`). Мы лишь приведем два примера ее использования.

Подготовим файл `exampl5.mp` такого содержания:

```
beginfig(1)
path a,b,c;
a = halfcircle rotated -90 scaled 4cm;
b = fullcircle scaled 4cm  shifted (2.6cm,0);
c = buildcycle(a,b);
fill c withcolor 0.9 white;
draw a; draw b;
picture l;
l=thelabel (btex $a\cap b$ etex, (1.3cm,0));
unfill bbox l;
draw l;
endfig;
bye
```



Затем выполним команду

```
mpost exampl5.mp
```

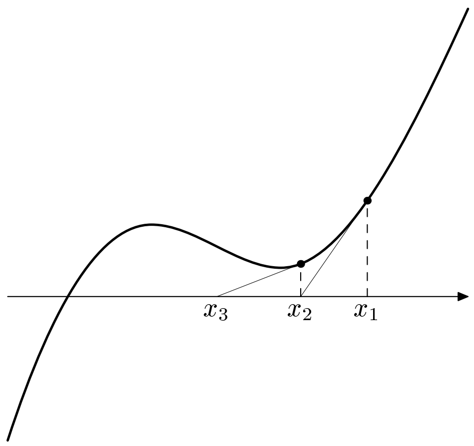
Получится файл `exampl5.1`. Его можно включить в текст с помощью команды `\includegraphics` как PostScript-файл (строго говоря, он не является полноценным PostScript-файлом, но `dvips` умеет его включать). Соответствующий рисунок изображен справа от исходного текста `metapost`-программы.

А вот еще один пример использования системы `metapost` (заимствован из авторского руководства по этой системе):

```
beginfig(1);

3.2scf = 2.4in;
```

```
path fun;
# = .1;  % Keep the function single-valued
fun = ((0,-1#)..(1,.5#){right}..(1.9,.2#){right}..{curl .1}(3.2,2#))
  scaled scf yscaled(1/#);

vardef vertline primary x = (x,-infinity)..(x,infinity) enddef;
primarydef f atx x = (f intersectionpoint vertline x) enddef;
primarydef f whenx x = xpart(f intersectiontimes vertline x) enddef;

z1a = (2.5scf,0);
z1 = fun atx x1a;
y2a=0; z1-z2a=whatever*direction fun whenx x1 of fun;
z2 = fun atx x2a;
y3a=0; z2-z3a=whatever*direction fun whenx x2 of fun;

draw fun withpen pencircle scaled 1pt;
drawarrow (0,0)..(3.2scf,0);

label.bot(btex $x_1$ etex, z1a);
draw z1a..z1 dashed evenly;
makelabel(nullpicture, z1);
draw z1..z2a withpen pencircle scaled .3;
label.bot(btex $x_2$ etex, z2a);
draw z2a..z2 dashed evenly;
makelabel(nullpicture, z2);
draw z2..z3a withpen pencircle scaled .3;
label.bot(btex $x_3$ etex, z3a);
endfig; bye
```

Этот рисунок иллюстрирует метод Ньютона поиска корня уравнения $f(x) = 0$; заметим, что направления касательных и точки пересечения вычисляются автоматически.



5. Зеркальный вывод и кресты

При подготовке оригинал-макета для типографии часто возникает необходимость вывести текст (на пленки) в зеркальном изображении. Если вы пользуетесь драйвером `dvips`, то это удобно сделать с помощью стилевого пакета `crop`. Именно, если подключить пакет `crop` с опцией `mirror` (т. е. если записать в преамбуле документа строку `\usepackage[mirror]{crop}`), то в PostScript-файле, полученном после обработки `dvi`-файла программой `dvips`, все страницы будут записаны в зеркальном изображении, и именно так они напечатаются, если послать это PostScript-файл на принтер. Подчеркнем, что при использовании `dvi`-драйверов, отличных от `dvips`, этот прием не сработает.

При использовании пакета `crop` открывается также возможность напечатать «кресты», помогающие при монтаже оригинал-макета в типографии; по поводу того, как это сделать, см. комментарии в файле `crop.sty`, который, собственно говоря, и представляет собой стилевой пакет `crop`.

6. PostScript-шрифты

Последнее, о чем мы хотим сказать в связи с языком PostScript, — это PostScript-шрифты. Фирма Adobe разрабатывала язык PostScript применительно к нуждам полиграфии. Естественно, что она параллельно изготовила описания сотен (если не тысяч) шрифтов в формате, понятном PostScript'у. Такие шрифты называют еще «Type 1»-шрифтами (строго говоря, есть и другие форматы шрифтов, доступные PostScript'у, но большинство шрифтов имеет формат Type 1). Часто такие шрифты хранятся в файлах с расширениями `pfa` или `pfb`.

Вслед за фирмой Adobe множество других фирм изготовили тысячи шрифтов (оригинальных, а также напоминающих классические образцы литых шрифтов или шрифтов фотонабора). Возникает естественное желание использовать эти шрифты с системой TEX. Что для этого нужно? Об этом (и о возникающих трудностях) пойдет речь в приложении В.

Приложение B

Шрифты и LaTeX

1. Гарнитуры

Если при чтении книжек вы больше интересовались содержанием, чем оформлением, то могли и не заметить, что они набираются разными шрифтами. Здесь мы имеем в виду не шрифтовые выделения (скажем, жирный — или, как сказали бы полиграфисты, **полужирный** шрифт), а различные рисунки шрифтов. Полиграфисты называют их *гарнитурами.* Заглянув в выходные данные книги (особенно изданной несколько десятилетий назад), можно найти там слова «Гарнитура обыкновенная новая» или «Литературная гарнитура». Эти слова означают разные рисунки букв; каждая из гарнитур может включать в себя шрифты различных начертаний (прямые, курсивные, полужирные) и размеров.

Вот несколько образцов различных гарнитур:

This is Times font sample:
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

This is Palatino font sample:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

This is Avant Garde font sample:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

This is Helvetica font sample:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

This is Courier font sample:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Вначале показаны четыре начертания гарнитуры Times[1] — прямое светлое, курсивное светлое, прямое полужирное и курсивное полужирное. Аналогичные варианты есть и для других гарнитур нашего примера, но для экономии места они не показаны. Кроме того, не показаны (также входящие в число символов) цифры, знаки препинания и т. п.

2. Что нужно LaTeX'у от шрифта?

Допустим, нам понравилась какая-то гарнитура и мы хотим использовать такой (точнее говоря, очень похожий) на нее шрифт в системе LaTeX. Чтоб для этого нужно?

Прежде всего нужно иметь файлы, описывающие формы букв выбранной нами гарнитуры. Проще всего, если эти файлы подготовлены в формате METAFONT (как это сделано Кнутом для шрифтов Computer Modern). Тогда с помощью программы METAFONT из них можно получить `tfm`-файлы с описанием размеров букв.

Немного сложнее обстоит дело со шрифтами в формате Type 1. Но и для этого формата написаны программы, позволяющие полуавтоматически изготавливать `tfm`-файлы (основная проблема, требующая ручного вмешательства, — несовпадение кодировок).

После этого можно попросить LaTeX использовать эти файлы вместо обычно используемых им `tfm`-файлов для шрифтов Computer Modern, и получить `dvi`-файл. (Подробнее об этом см. ниже). Но чтобы использовать этот файл, `dvi`-драйвер должен либо иметь `pk`-файлы с матрицами букв, либо уметь использовать вместо них описания шрифтов в формате Type 1 (и иметь эти описания). Обе возможности встречаются на практике. Например, программа `xdvi` (показывающая `dvi`-файлы на экране в X Window System под UNIX) использует `pk`-шрифты. Эти шрифты программа `xdvi` получает из шрифтов в формате Type 1 с помощью программы `gsftopk`. А программа `dvips` просто вставляет

[1] Строго говоря, нельзя называть эту гарнитуру «Times», поскольку она лишь очень напоминает гарнитуру Times, но во избежание юридических проблем так не называется. Мы опускаем подобные оговорки в дальнейшем.

в результирующий PostScript-файл описания соответствующих шрифтов в формате Type 1. (При этом используется механизм «виртуальных шрифтов», но его описывать мы не будем.)

При этом получающийся файл имеет то достоинство, что он не зависит от разрешения: в нем содержатся не матрицы букв того или иного размера (как будет с pk-шрифтами), а векторные описания форм букв. Такие PostScript-файлы лучше выдерживают увеличение.

Для примера посмотрите на надпись «Это строка (string)!» в приложении Б. При изготовлении соответствующего PostScript-файла были использованы pk-шрифты разрешения 600 dpi. Затем эта надпись была включена в текст с увеличением вдвое, поэтому в итоговом оригинал-макете она соответствовала разрешению 300 dpi и выглядела хуже, чем такая же надпись в разрешении 600 dpi. Такого не случается с векторными шрифтами: сравните с надписью «This is a string», которая использует (предусмотренные стандартом PostScript) векторные шрифты. Впрочем, заметить разницу в плавности линий можно только на хорошем принтере и при внимательном разглядывании.

3. Добавление одного шрифта

ПУСТЬ теперь у нас есть `tfm`- и `pk`-файлы для какого-то шрифта (и они находятся в тех местах, где TeX их ищет). Как добавить такой шрифт в наш текст? Самый простой и самый грубый способ добавить новый шрифт предоставляет команда `\newfont`. Ее формат таков:

`\newfont{`*команда*`}{`*описание_шрифта*`}`

Здесь команда — это придуманная вами команда для переключения на добавляемый вами шрифт. Имя этой команды надо выбирать так же, как первый аргумент команды `\newcommand` (оно не должно быть занято, не должно начинаться на `end` и т. п.). Что же до *описания_шрифта,* то в простейшем виде это — просто имя `tfm`-файла, соответствующего данному шрифту.

Вот пример. В свое время Дональд Кнут шутки ради разработал причудливый шрифт, называемый `cmff10` (его `tfm`-файл и `mf`-файл называются соответственно `cmff10.tfm` и `cmff10.mf`, и входят в стандартный комплект). Чтобы пользоваться этим шрифтом в своем тексте, включите в преамбулу строку

```
\newfont{\weird}{cmff10}
```

и вы сможете писать тексты вроде

The letters look strange.	`The letters look {\weird strange.}`

Команду, определенную с помощью `\newfont`, можно использовать только так, как в этом примере, а именно, переключаться с ее помощью на новый шрифт внутри группы, и в этом отношении она похожа на команды переключения шрифтов наподобие `\itshape` или `\bfseries`. На этом, однако, сходство и кончается: `\itshape` или `\bfseries` меняют начертание шрифта, но сохраняют его размер, в то время как команда, определенная с помощью `\newfont`, всегда выдает шрифт одного размера — того, что записан в `tfm`-файле, на который вы ссылаетесь в ее втором аргументе. Далее, некоторые LaTeX'овские команды для переключения шрифтов (именно, команды, меняющие размер) не только переключаются на шрифт другого размера, но и меняют интерлиньяж (а также, кстати, размер невидимой линейки, создаваемой командой `\strut`); команда, определенная с помощью `\newfont`, никакого влияния ни на интерлиньяж, ни на смысл команды `\strut` не окажет. Наконец, внутри математической формулы такая команда вообще будет проигнорирована.

Тем не менее, если вы нашли какой-то экзотический символ, который хотите несколько раз использовать, и нашли шрифт, в котором он есть, команда `\newfont` может быть полезна. Например, в шрифте `wasy10`, входящем во многие комплекты TeX'а, в позиции 7 имеется символическое изображение телефона. Зная это, можно написать в преамбуле

```
\newfont{\wasyten}{wasy10}
\newcommand{\telephone}{\hbox{\wasyten\symbol{7}}}
```

и использовать команду `\telephone` для изображения телефона (☎).

Еще одно применение этой команды — масштабирование шрифтов. Чтобы подключить масштабированный шрифт с помощью команды `\newfont`, надо задать требуемое увеличение или уменьшение во втором аргументе команды `\newfont`. Оно задается с помощью TeX'овского «ключевого слова» `scaled` (без backslash'а!), за которым следует коэффициент масштабирования, умноженный на 1000 (после умножения коэффициента на 1000 должно получиться целое число). Например, для подключения шрифта, увеличенного в два с половиной раза, надо после имени `tfm`-файла написать `scaled 2500`, а для шрифта, размеры которого уменьшены на 30%, надо написать `scaled 700`.

Можно также задавать увеличение не в явном виде, а сообщить TeX'у требуемый «характерный размер» шрифта. Для этого надо во втором аргументе команды `\newfont` написать после имени `tfm`-файла масштабируемого шрифта

`at` *размер*

где *размер* — требуемый «характерный размер», заданный обычным образом в TeX'овских единицах длины или через TeX'овские параметры длины, а `at` — еще одно TeX'овское «ключевое слово» (пишущееся без backslasha'a). Если основной шрифт документа имеет кегль 10, то характерный размер разумно выбирать равным `10pt`, если 11 или 12, то `11pt` или `12pt`.

Но, конечно, это зависит от ситуации. Скажем, можно написать

```
\newfont{\wasytwenty}{wasy10 at 20pt}
\newcommand{\bigtelephone}{\hbox{\wasytwenty\symbol{7}}}
```

и получить вдвое больший рисунок телефона: ☎

Еще одно применение масштабирования можно увидеть в начале этого раздела, где с его помощью изготовлен инициал «П». При этом использован такой код:

```
\newfont{\initial}{zcr17 at 48pt}
\newbox{\literaP}
\savebox{\literaP}{\hbox{\initial П}}
\begin{wrapfigure}{l}{0.75\wd\literaP}
\vbox to 0.4\ht\literaP{%
   \vss \usebox{\literaP}%
   \vspace*{-0.2\ht\literaP}%
}%
\end{wrapfigure}
УСТЬ теперь...
```

4. Семейства шрифтов

Конечно, хотелось бы подключать к LaTeX'у не один шрифт фиксированного размера, а целый набор новых шрифтов таким образом, чтобы в `tex`-файле можно было менять их начертание и размер с помощью команд, аналогичных рассмотренным нами в разд. 5.2 главы III.

Такие возможности действительно есть. Например, образцы шрифтов типа Times, приведенные на с. 360, были получены с помощью таких команд:

```
{\fontfamily{ptm}
\selectfont
\begin{center}
This is Times font sample\\
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz\\
```

```
\fontshape{it}\selectfont
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
\fontseries{b}\fontshape{n}\selectfont
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz\\
\fontshape{it}\selectfont
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
\end{center} }
```

В них устанавливаются желаемые параметры шрифта (см. разд. III.5.2), а затем этот шрифт вызывается командой `\selectfont`.

Сначала мы устанавливаем семейство шрифтов с помощью команды

```
\fontfamily{ptm}
```

(где `ptm` есть внутреннее имя семейства шрифтов типа таймс). При этом остальные параметры (размер, насыщенность, начертание) остаются неизменными. Команда `\selectfont` переключает шрифт. Затем мы меняем начертание с помощью команды `\fontshape`, делая его курсивным (`it`, прямое начертание обозначается `n`), и снова переключаем шрифт с помощью `\selectfont`. После этого мы меняем насыщенность с помощью команды `\fontseries` (при этом `b` означает полужирный шрифт, а обычный шрифт обозначается `m`) и возвращаемся к прямому начертанию. И так далее.

Есть еще команда изменения размера шрифта, явно указывающая кегль и интерлиньяж. Например, команды

```
\fontfamily{ptm} \fontsize{17pt}{24pt}
\selectfont This is Times Roman\\ 17 pt / 24 pt
```

позволяют напечатать две строки шрифтом типа таймс с размером 17 pt и расстоянием между строками 24 pt:

This is Times Roman

17 pt / 24 pt

Команды `\fontfamily`, `\fontseries`, `\fontshape` и `\fontsize` являются командами «нижнего уровня» (подробнее о них см. в разд. 6.3): как правило, надо использовать не их, а команды типа `\textbf`, `\textit`, `\textsf` или же команды смены начертания и насыщенности, описанные в разд. III.5.2 (`\rmfamily`, `\sffamily`, `\upshape`, `\bfseries` и др.).

Но откуда же LaTeX узнаёт, что, например, команда `\rmfamily` предписывает переключиться на семейство шрифтов, известное под внутренним именем `cmr`, а команда `\bfseries` устанавливает насыщенность, известную под внутренним именем `bx`? Эта информация содержится в определении следующих команд:

Команда	Определение в стандартных классах
`\rmdefault`	`cmr`
`\sfdefault`	`cmss`
`\ttdefault`	`cmtt`
`\bfdefault`	`bx`
`\mddefault`	`m`
`\itdefault`	`it`
`\sldefault`	`sl`
`\scdefault`	`sc`
`\updefault`	`n`

Читать эту таблицу надо следующим образом: если семейство, насыщенность или начертание шрифта обозначается буквами `xx` (как в разд. III.5.2), то команда `\xxfamily`, `\xxseries` или `\xxshape` (или команда с одним аргументом `\textxx`) установит соответствующий атрибут шрифта с внутренним именем, указанным в таблице в строчке с командой `\xxdefault`.

Сказанное объясняет рецепт уменьшения ширины жирного шрифта, приведенный на с. 114: переопределяя команду `\bfdefault`, мы заставляем LaTeX по командам `\textbf` или `\bfseries` (а также `\bf`) выбирать шрифт насыщенности `b`, менее широкой, чем насыщенность `bx`, получающаяся по умолчанию.

А если, например, вы захотите, чтобы основным шрифтом вашего документа был шрифт без засечек, то можно переопределить в преамбуле команду `\rmdefault`:

```
\renewcommand*{\rmdefault}{cmss}
```

(согласно нашей таблице, `cmss` — это LaTeX'овское внутреннее имя для шрифта без засечек). Другой вопрос, будет ли это удачным полиграфическим решением. Не слишком увлекайтесь подобными экспериментами, если вы не полиграфист!

5. Еще о русских шрифтах: подробности для знатоков

Сейчас мы сообщим вам две новости: хорошую и плохую. Хорошая состоит в том, что большая часть из сказанного выше совершенно не нужна для пользования различными шрифтами в LaTeX'е, поскольку разработчики стилевых пакетов о вас позаботились. Скажем, если организаторы конференции просят присылать статьи в гарнитуре Times, достаточно добавить в преамбулу команду

```
\usepackage{times}
```

После этого все шрифты заменятся на шрифты типа таймс без каких-либо дополнительных усилий с вашей стороны. (Правда, могут появиться новые overfull'ы, так как размеры букв другие). Эта команда не меняет шрифтов, используемых в формулах, и они выглядят не очень хорошо (гарнитура таймс в среднем жирнее обычной Computer Modern). Чтобы изменить и шрифты для формул, скажите

```
\usepackage{mathptmx}
```

Более подробно смена шрифтов в тексте и формулах обсуждается в книге [6]; учтите только, что соответствующие пакеты часто меняются.

Теперь плохая новость: все перечисленные удобства (возможности легко использовать различные гарнитуры) *доступны лишь при использовании латинских шрифтов*. Если вы попробуете добавить русские буквы к нашим примерам, то ничего хорошего не получится. Прежде всего это связано с нехваткой свободно распространяемых русских шрифтов различных гарнитур, а также с различиями в кодировках русских символов. Этим бедам и посвящен настоящий раздел, в котором автор попытался кратко изложить известную ему информацию про положение дел с русскими шрифтами. Вряд ли она сильно поможет многим на практике, но если она побудит кого-либо из читателей взяться за дело и исправить положение (хотя бы немного), то автор будет считать свою задачу выполненной. Автор заранее приносит извинения тем (вероятно, многочисленным) коллегам, о работах которых у него нет достаточной информации; вина, разумеется, тут только его.

Для использования разнообразных русских шрифтов в LaTeX'е нужно решить две проблемы: (а) найти сами шрифты и (б) написать файлы для подключения их к LaTeX'у. В этом разделе мы обсудим первую из этих проблем, а второй из них посвятим следующий раздел.

Нас будут интересовать шрифты двух типов: METAFONT-шрифты и PostScript-шрифты. Еще бывают так называемые TrueType шрифты, их подключение к TeX'у в принципе возможно, но менее удобно, и о них мы говорить не будем.

5.1. METAFONT-шрифты.

Эти шрифты в свою очередь делятся на две категории. В первом случае авторы ставили своей целью экстраполировать Computer Modern на русские буквы (и, в частности, заимствовали формы общих букв из Computer Modern). Таковы шрифты AMS, шрифты Глонти и шрифты Лапко (см. ниже) Во втором случае авторы старались сделать русский шрифт определенного вида, не связывая себя требованиями единства стиля с Computer Modern.

В комплект шрифтов AMS (Американского математического общества), распространяемый практически со всеми версиями TeX'а, входят шрифты, разработанные в Humanities and Arts Computing Center, University of Washington (в 1989 году). Эти шрифты использовались в различных публикациях AMS (в основном — для печати русских названий статей и книг в списках литературы). Вот образец шрифта `wncyr10`:

АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Эти шрифты, если можно так выразиться, «импортные». Первыми «отечественными» русскими METAFONT-шрифтами, кажется, были шрифты, разработанные в Институте физики высоких энергий (Протвино) при участии Наны Глонти и Александра Самарина (их иногда называют шрифтами Глонти). Вот образец шрифта `cmcyr10` (взят с сервера `ftp.dante.de`)

АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Эти шрифты (с небольшими изменениями отдельных букв, сделанными в основном А. Шенем) были использованы при подготовке этой книги. Приведем их (с этими изменениями) в том же масштабе:

АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Наконец, Ольга Лапко и Андрей Ходулев разработали шрифты, получившие название «LH-шрифтов». Многие считают, что из различных METAFONT-продолжений шрифта Computer Modern именно эти шрифты обладают наиболее высоким качеством. Кроме того, они включают в себя большой набор символов самых разных языков на кириллической основе. Можно надеяться, что они в будущем окончательно стабилизируются (пока что новые версии выходят довольно часто) и будут в разных версиях TEX'а столь же стандартны, как сейчас шрифты Кнута. (Уже сейчас LH-шрифты входят в большинство дистрибутивов TEX'а, обычно в не самых последних версиях. Впрочем, дело осложняется тем, что авторами этих шрифтов предусмотрено много разных версий для разных кодировок, причем они генерируются «на лету» с использованием довольно хитроумных и не везде работающих механизмов.)

Вот как выглядят русские буквы в этих шрифтах:

АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдёежзийклмнопрст
уфхцчшщъыьэюя

Разница между этими четырьмя образцами заметна, если приглядеться внимательно. (Обратите внимание на буквы «Л», «Д» и «Э».)

К шрифтам второй категории относятся шрифты, авторы которых не стремились следовать стилю Computer Modern.

При поддержке Российского фонда фундаментальных исследований в рамках проекта «Русский TEX» Л. Н. Знаменская и С. В. Знаменский разработали шрифты, отчасти напоминающие гарнитуру «Обыкновенная новая». Вот образцы этих шрифтов (мы воспроизводим предварительный вариант этих шрифтов, вошедший, с разрешения разработчиков, в использованную при подготовке книги русификацию):

Образцы шрифтов типа «Обыкновенной новой гарнитуры»
на основе METAFONT-файлов Знаменских

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ

абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Видно, что общие для латинского и русского алфавита буквы имеют несколько разные начертания (латинские буквы не изменились по сравнению с оригинальными шрифтами Computer Modern Кнута)

Были предприняты также попытки изготовить METAFONT-файлы на основе других популярных гарнитур. Часть полученных таким образом шрифтов также вошли в использованную при подготовке этой книги русификацию. Вот образцы этих шрифтов:

Образцы шрифтов типа таймс (А. Шень, М. Ушаков)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Образцы шрифтов типа «гельветика» (О. Зорина, А. Шень)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
АБВГДЕЁЖЗИЙКЛМНОПРСТ
УФХЦЧШЩЪЫЬЭЮЯ
абвгдеёжзийклмнопрст
уфхцчшщъыьэюя

Завершая разговор о METAFONT-шрифтах, отметим, что существует (не вполне завершенный) набор программ `ps4mf` (подробную информацию можно найти на сервере `ftp.dante.de`), который позволяет полуавтоматически изготавливать METAFONT-шрифты из Type 1 шрифтов. Однако получающиеся шрифты плохо приспособлены к печати при низких и средних разрешениях (например, для 300 dpi). Дело в том, что и METAFONT, и Type 1 предусматривают задание границ букв в виде кусочно-гладких кривых, и эти кривые легко перевести из одного формата в другой с минимальными изменениями. Проблема в том, что при переводе шрифтов в черно-белую матрицу кривые подвергаются округлению с неизбежными погрешностями. Если, скажем, ширина вертикальной палочки в букве «Ш» не есть точное кратное размера точки, то может случиться, что три палочки (которые имеют одну и ту же ширину в точном представлении) при округлении приобретут разную ширину. Кроме того, из-за того же округления буква «Ш» может потерять осевую симмметрию, что нежелательно. Эта проблема существует и для METAFONT'а, и для Type 1, но решается по-разному. Язык METAFONT предоставляет автору шрифтов возможности самому влиять на округление так, как он этого пожелает. В шрифтах Type 1 используются указания особого вида (hints), которые побуждают интерпретатор PostScript'а производить округления так, как это хотелось автору шрифта. При этом формат и интерпретация этих указаний — дело довольно сложное, и без потерь в METAFONT они не переводятся.

Обратный перевод (из METAFONT'а в Type 1) сталкивается не только с проблемой округления, но и с тем, что в METAFONT'е (который представляет собой язык программирования) есть разнообразные средства описания шрифтов, а не только описание контуров. На практике, изготавливая Type 1 шрифты из METAFONT-шрифтов, сначала изготавливают матрицы букв в высоком разрешении, после чего применяют программы, восстанавливающие границы кривых, и добавляют указания (hints) вручную или полуавтоматически.

5.2. PostScript-шрифты.

В отличие от кириллических METAFONT-шрифтов, которые можно пересчитать по пальцам, PostScript-шрифты с русскими буквами существуют в изобилии. Некоторые из них распространяются западными производителями (скажем, той же фирмой Adobe), но подавляющее большинство сделано в России. Этим, в частности, много занималась фирма «Параграф» (впоследствии шрифтовое подразделение фирмы получило название Paratype).

В отличие от METAFONT-шрифтов, которые (как правило) являются свободно распространяемыми, PostScript-шрифты надо покупать и нельзя выкладывать на сервер вместе с TEX-системой. Формально говоря, это не запрещает распространять вспомогательные файлы, необходимые для их подключения к TEX, как это делалось в течение многих лет с английскими PostScript-шрифтами. В конце концов, наиболее законопослушные пользователи могут шрифты купить[2], а менее

[2]Впрочем, не всё так просто. Например, посмотрев на сайт `www.paratype.com`, можно обнаружить разные типы лицензий. Одни ограничивают число принтеров, на которых можно печатать полученные файлы (тем самым, скажем, переслать статью

законопослушные пойдут в ближайший ларек за «пиратским» компакт-диском. Но и в том, и в другом случае нет никаких гарантий, что удастся достать именно ту версию шрифта, что была у разработчика вспомогательных файлов — русские шрифты еще менее стандартны, чем английские, и не только формы букв, но и их размеры и кодировка могут различаться в разных версиях.

В случае латинских шрифтов ситуация сильно упростилась благодаря фирме URW, которая разрешила распространять бесплатно некоторые наиболее популярные шрифты вместе с системой `Ghostscript`. Именно эти шрифты и были использованы нами в приведенных выше примерах гарнитур на с. 360. К сожалению, насколько известно автору, этому примеру никто из фирм-производителей кириллических шрифтов не последовал, и «белых» (с ясным происхождением и свободно распространяемых без ограничений на использование при подготовке печатных и электронных документов) качественных PostScript-шрифтов автору найти не удалось.[3] Но есть некоторые «серые» шрифты:

- Давным-давно фирма Urbansoft распространяла версию системы Linux, которая называлась «Открытое ядро» (занимавшую тогда несколько дюжин пятидюймовых дискет). В ее состав входили некоторые PostScript-шрифты с русскими буквами (в том числе комплект шрифтов типа литературной гарнитуры), однако они не содержали внутри себя сведений о копирайте. Возможно, какие-то из этих шрифтов послужили исходным материалом для пакета PSCyr (см. следующий пункт).

- В состав нескольких недавних версий Linux входит пакет PSCyr. Один из разработчиков этого пакета, Александр Лебедев, пишет (цитируется по версии 0.4c-patch2, взятой с сервера `ftp.vsu.ru`): «*Первые шрифты в коллекцию русских шрифтов* ⟨...⟩ *были собраны Сергеем Наумовым в 1995 г. В этих шрифтах отсутствовала информация об изготовителе, однако по некоторым признакам это были просканированные изображения широко распространенных русских типографских шрифтов...*

  В 1997 г. коллекция была дополнена шрифтами Arial, Courier и

с этими шрифтами коллегам уже нельзя), другие (лицензия на одну публикацию) требуют покупать отдельную лицензию на каждую статью и т. п. Вряд ли даже законопослушные пользователи станут широко использовать шрифты с такими лицензиями.

[3] Недавно появились предназначенные для свободного распространения шрифты В. Филиппова, дополняющие упомянутые выше URW-шрифты русскими буквами; видимо, работа над ними еще не закончена. См. `ftp://ftp.gnome.ru/fonts/urw`.

Times фирмы Monotype, построенными из шрифтов стандартной поставки Windows...

Свободно распространяемый шрифт ERKurier ⟨...⟩ был добавлен Константином Чумаченко в версии 0.2, а шрифт Academy (неизвестного происхождения) — им же в версии 0.3.».

Кроме того, говоря об изменениях в версии 0.3, Лебедев пишет: *«для легализации коллекции шрифтов добавлен файл LICENSE»*. В этом файле говорится: *«Permission is granted to use, copy, and freely redistribute the fonts from the PSCyr font collection in any media... provided that ⟨...⟩ 1. any modification of* `*.afm` *and* `*.pfb` *files is forbidden. 2. the redistribution of the font(s) and the whole collection as separate computer file(s) should retain this license information»*.

В самих шрифтах встречаются: Monotype Corporation, AG Fonts Collection, Andrejs Grinbergs, Gavin Helf, Ares Software.

В файле README к версии 0.3 написано: «*The typefaces included in this package originally come from The Monotype Corporation plc. (with changes by Eugene V. Demidov), Gavin Helf (with changes by Andrey Chernov); Paratype, division of Paragraph International. As to my knowledge, these fonts are placed into public domain and can be freely redistributed. (AUTHOR: Konstantin Chumachenko)*».

Вопрос о том, насколько законно объявлять свободно распространяемыми шрифты неизвестного происхождения, не указывая никаких подтверждений того, что первоначальные разработчики поместили их в public domain, сложен, тем более что законы, касающиеся юридической защиты шрифтов, непросты для понимания и различны в разных странах. Мы условно относим эти шрифты к категории «серых», имея в виду лишь, что вопрос о законности их распространения кажется сложным. Впрочем, в условиях, когда на уличных лотках можно найти по бросовой цене базы данных государственных организаций и новейшие версии Windows, трудно представить себе, что именно эти шрифты станут темой юридических дебатов.

- Наиболее понятна ситуация со шрифтами Литературной гарнитуры, разработанными фирмой Параграф (Paratype) по заказу Российского фонда фундаментальных исследований. (Заметим, что эта гарнитура отличается от других версий Литературной гарнитуры той же фирмы). По замыслу РФФИ, этот заказ должен был дать возможность готовить и распространять научные тексты, сверстанные в одной из наиболее распространенных традиционных гарнитур. К сожалению, прилагаемая к пакету лицензия не

> так проста для понимания. Там написано: «*In the following paragraphs “fonts” stands for Type 1 fonts together with metric files and also any conversions and renderings of them. You are allowed: to use fonts with any TeX distribution; to use fonts with any graphic editors for preparing pictures to be included with TeX documents; to use fonts with other programs (such as WWW browsers) for screening purpose only; to convert fonts into any format for above purposes; to include fonts into electronic documents (such as PDF); to distribute fonts together with RFBR TeX distribution.*»
>
> Казалось, бы все возможные разумные использования тут предусмотрены и разрешены. Однако далее написано, что «*You are not allowed to use fonts for hardcopy making with any programs except TeX drivers*», что, формально говоря, запрещает печать включающего эти шрифты файла (полученного с помощью системы TeX) с использованием (скажем) программ `Adobe Acrobat` или `Ghostscript`. Еще написано, что нельзя «*distribute this fonts as an independent package*» (так в оригинале), причем что такое в точности RFBR TeX distribution, с которой распространять можно, и где ее взять, не вполне ясно.
>
> Можно надеяться, что это лишь нечеткость формулировки и что дух лицензии важнее ее буквы.

Говоря об отсутствии «белых» кириллических PostScript-шрифтов, следует внести уточнение. Существуют совершенно легальные свободно распространяемые PostScript-шрифты в пакете CM-Super (автор — Владимир Волович), полученные преобразованием свободно распространяемых METAFONT-шрифтов (включая LH-шрифты, шрифты Computer Modern и другие). Но с точки зрения расширения ассортимента шрифтов этот пакет не особенно полезен, поскольку все входящие в него шрифты есть и в METAFONT-формате. Польза от него в другом: с его помощью можно готовить в системе TeX PostScript- и PDF-файлы, в которых использованы только шрифты Type 1 (такие файлы лучше приспособлены к просмотру в программе `Acrobat Reader` и к печати на принтерах с высоким разрешением).

Шрифты эти были получены так: сначала из METAFONT-текстов были получены матрицы высокого разрешения, а затем они преобразованы в формат Type 1 с помощью программ `TeXtrace` и `Autotrace`.

6. Подключение шрифтов

До сих пор мы говорили о наличии шрифтов как таковых, а не о механизме их подключения к системе LaTeX. Само по себе подключение не очень сложно, и можно надеяться, что если качественные разнообразные свободно распространяемые METAFONT- или PostScript-шрифты появятся, то вскоре будут написаны и LaTeX-определения, их подключающие.

Тем не менее, скажем несколько слов о том, как работает механизм подключения шрифтов.

6.1. Понятие кодировки. Пример: кодировка T1.

Система LaTeX пытается реализовать следующую простую идею: набор символов и форма символов (гарнитура) являются независимыми координатами. Скажем, можно говорить о русских и латинских шрифтах (два значения первой координаты), а также о шрифтах гарнитуры таймс и гельветика (два значения второй координаты), и двигаться по каждой координате независимо.

Несмотря на кажущуюся очевидность, этот принцип трудно реализовать буквально, и не только потому, что какие-то значения координат соответствуют отсутствующим в данном комплекте шрифтам, но и по более тонким причинам. Например, в какой-то гарнитуре может быть специальный символ «fi», которым изображаются стоящие рядом буквы f и i (это называется «лигатурой», обратите внимание на отличие этого символа от набранных подряд букв: fi), а в другой гарнитуре соединять эти буквы не принято и такого символа нет.

Тем не менее в первом приближении можно считать, что в системе LaTeX есть два независимых параметра: кодировка (encoding) говорит, какие символы имеются, а семейство (font family) определяет внешний вид символов. Мы уже упоминали кодировку T1, используемую при работе с неанглийскими языками на латинской основе. Таблица В.1 показывает символы кодировки T1, представленные шрифтами семейства Computer Modern. Та же самая кодировка T1, представленная шрифтами семейства Helvetica, показана в таблице В.2. В теории эти две таблицы должны отличаться лишь формой символов, но на практике это не совсем так: в нынешней версии LaTeX'а некоторые позиции в кодировке T1 в шрифте типа Helvetica не заполнены соответствующими буквами.

Тем не менее кодировка T1 (ее еще называют Cork encoding, поскольку кодировка была утверждена на конференции TeXников в ирландском городе Корк) обеспечивает более или менее приемлемую основу для использования LaTeX'а с разными шрифтами и разными языками с латинской графикой.

Впрочем, до сих пор по умолчанию используется так называемая кодировка OT1, в которой всего 128 символов, представленных в ориги-

Таблица В.1. Кодировка T1: Computer Modern

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	‚	‹	›
"10	“	”	„	«	»	–	—		₀	ı	ȷ	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	ˆ	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	˜	-
"80	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	Ĺ	Ľ	Ł	Ń	Ň	Ŋ	Ő	Ŕ
"90	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	Ÿ	Ź	Ž	Ż	IJ	İ	đ	§
"A0	ă	ą	ć	č	ď	ě	ę	ğ	ĺ	ľ	ł	ń	ň	ŋ	ő	ŕ
"B0	ř	ś	š	ş	ť	ţ	ű	ů	ÿ	ź	ž	ż	ij	¡	¿	£
"C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
"D0	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS
"E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
"F0	ð	ñ	ò	ó	ô	õ	ö	œ	ø	ù	ú	û	ü	ý	þ	ß

нальных шрифтах Кнута. Эта кодировка показана в таблицах В.3 и В.4 в двух видах (обычные шрифты и `tt`-шрифты). Читатель может заметить, что эти таблицы не соответствуют идее кодировки, объясненной выше: далеко не все символы в двух таблицах соответствуют друг другу. (А в курсивных шрифтах на месте доллара появляется фунт!) Причина этого понятна: шрифты разрабатывались Кнутом исходя из ограничения в 128 символов и до появления общей схемы переключения шрифтов.

6.2. Входная и внутренняя кодировки.

Следует подчеркнуть, что слово «кодировка» в сочетании «кодировка T1» имеет не совсем обычный смысл. Эта кодировка, вообще говоря, не совпадает с кодировкой букв во входных файлах (а для некоторых символов вообще нет кода, который им соответствует во входном файле).

Таблица В.2. Кодировка T1: Helvetica

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	‚	‹	›
"10	“	”	„	«	»	–	—		■	ı	▮	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	-
"80	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	Ĺ	Ľ	Ł	Ń	Ň	■	Ő	Ŕ
"90	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	Ÿ	Ź	Ž	Ż	Ĳ	İ	đ	§
"A0	ă	ą	ć	č	ď	ě	ę	ğ	ĺ	ľ	ł	ń	ň	■	ő	ŕ
"B0	ř	ś	š	ş	ť	ţ	ű	ů	ÿ	ź	ž	ż	ĳ	¡	¿	£
"C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
"D0	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS
"E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
"F0	ð	ñ	ò	ó	ô	õ	ö	œ	ø	ù	ú	û	ü	ý	þ	ß

Как же задается соответствие между ними? Есть два принципиально разных подхода.

Первый вариант состоит в том, что в программу TeX встраивается дополнительный механизм (выходящий за рамки собственно TeX'а), который считывает таблицу соответствия между кодами и автоматически перекодирует входные символы. (Можно сказать, что TeX в узком смысле этого слова ничего не знает о входной кодировке.) Обратное перекодирование производится при выдаче `log`-файлов и других текстовых файлов.

Второй вариант, не выходящий за рамки TeX'а и применяемый в пакете `inputenc`, состоит в том, что символы входного потока становятся «активными», то есть становятся командами, порождающими символы во внутренней кодировке. Это, конечно, противоречит всей архитектуре TeX'а, поскольку там есть специальная «категория», называемая буквами (letters), а активные символы — уже не буквы. В результате

Таблица В.3. Кодировка OT1

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
"10	ı	ȷ	`	´	ˇ	˘	¯	˚	¸	ß	æ	œ	ø	Æ	Œ	Ø
"20	˗	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	¡	=	¿	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	“	]	ˆ	˙
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	–	—	˝	˜	¨

Таблица В.4. Кодировка OT1: семейство `cmtt` (ttfamily)

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	↑	↓	'	¡	¿
"10	ı	ȷ	`	´	ˇ	˘	¯	˚	¸	ß	æ	œ	ø	Æ	Œ	Ø
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	¨

работа других пакетов может нарушиться, если они не готовы к такому повороту событий, в `idx`-файл попадают не буквы, а команды для них (что нарушает нормальную работу программ для обработки `idx`-файлов) и т. д. Зато, скажем, кодировку можно переключать «на лету» в середине текста, если это зачем-то понадобится.

Кстати, внутренняя кодировка (при использовании так называемых виртуальных шрифтов) может не совпадать и с кодировкой, используемой в шрифте (будь то PostScript- или METAFONT-шрифт). Но в `tfm`-файлах используется как раз внутренняя кодировка.

Что же получается с русскими текстами? Для них есть несколько входных кодировок (koi8-r, cp1251, cp866) и несколько внутренних. Заметим, что при использовании одной внутренней кодировки не удается поместить в нее все кириллические символы, сохранив первую половину такой же, как в кодировке T1 (разных символов слишком много). Это было причиной появления кодировок T2A, T2B и T2C, в которых

Таблица В.5. Кодировка T2A

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	I	⟨	⟩
"10	“	”	⌢	̏	˘	–	—		₀	ı	ȷ	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	ˆ	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	˜	-
"80	Ґ	Ғ	Ђ	Ћ	Һ	Җ	Ҙ	Љ	Ї	Қ	Ҟ	Ҝ	Ӕ	Ң	Ҥ	Ѕ
"90	Ө	Ҫ	Ў	Ү	Ұ	Ҳ	Џ	Ҹ	Ҷ	Є	Ә	Њ	Ё	№	¤	§
"A0	ґ	ғ	ђ	ћ	һ	җ	ҙ	љ	ї	қ	ҟ	ҝ	ӕ	ң	ҥ	ѕ
"B0	ө	ҫ	ў	ү	ұ	ҳ	џ	ҹ	ҷ	є	ә	њ	ё	„	«	»
"C0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
"D0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
"E0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
"F0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

русские буквы стоят на одних и тех же местах, но вот дополнительные буквы разные. В таблицах В.5, В.6 и В.7 приведены соответствующие списки символов (набранные шрифтами LH — на сегодняшний день единственными кириллическими шрифтами, эти кодировки поддерживающими).

Представляет интерес также кодировка X2 (табл. В.8), в которой первая половина также используется для кириллических символов (в отличие от T2A, T2B и T2C, в ней присутствуют символы, необходимые для набора русских текстов в дореволюционной орфографии). Кроме этого, имеется кодировка OT2, соответствующая кодировке символов в русских шрифтах AMS, а также многие другие.

6.3. Подключение шрифтов и fd-файлы.

Итак, мы знаем, что имеются пять (вообще говоря) независимых атрибутов шрифта. Все эти атрибуты можно увидеть в сообщении об over-

Таблица В.6. Кодировка T2B

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	I	⟨	⟩
"10	“	”	⌢	˵	˘	–	—		₀	ı	ȷ	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	˜	-
"80	Ӻ	Ғ	Г̡	Ҕ	Һ	Җ	δ	Ӡ	Љ	Қ	Ԯ	Ӄ	Ԓ	Ң	Ҥ	Ӈ
"90	Ө	Ҫ	Ў	Ү	Ӿ	Ҳ	Ӽ	Ҷ	Ӌ	Њ	Ә	Ԑ	Ё	№	¤	§
"A0	ӻ	ғ	г̡	ҕ	h	җ	δ	ӡ	љ	қ	ԯ	ӄ	ԓ	ң	ҥ	ӈ
"B0	ө	ҫ	ў	ү	ӿ	ҳ	ӽ	ҷ	ӌ	њ	ә	ԑ	ё	„	«	»
"C0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
"D0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
"E0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
"F0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

full'е (см. с. 116). Именно, там присутствует последовательность символов `\OT1/cmr/m/n/10.95`, в которой перечислены, через косую черту, атрибуты текущего шрифта (кодировка, семейство, насыщенность, начертание, размер). Их можно менять независимо.

- Кодировка задается с помощью команды `\fontencoding`, единственным аргументом которой является имя кодировки. Пример: `\fontencoding{T1}`. Иногда в ответ на команду `\fontencoding` LaTeX может пожаловаться, что кодировка ему неизвестна. Тогда можно дать предварительно команду `\DeclareFontEncoding` таким, например, образом:

  ```
  \DeclareFontEncoding{T2C}{}{}
  ```

  (две пары пустых фигурных скобок — не опечатка!). Может, конечно, случиться, что вы пытаетесь заказать кодировку, вообще

Таблица В.7. Кодировка T2C

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	I	⟨	⟩
"10	“	”	̑	‶	˘	–	—		0	ı	ȷ	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	ˆ	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	˜	-
"80	Ԥ	Ꚍ	Ҭ	Ҕ	Һ	Ҏ	Ҏ	Ӡ	Ӎ	Қ	Ԓ	Ҟ	Ӆ	Ң	Ӎ	Ӈ
"90	Ө	Ҽ	Ҿ	Ѣ	Ҋ	Ҳ	Џ	Ԛ	Ҷ	Ӊ	Ә	Ҏ	Ё	№	¤	§
"A0	ԥ	ꚍ	ҭ	ҕ	h	ҏ	ҏ	ӡ	ӎ	қ	ԓ	ҟ	ӆ	ң	ӎ	ӈ
"B0	ө	ҽ	ҿ	ѣ	ҋ	ҳ	џ	ԛ	ҷ	ӊ	ә	ҏ	ё	„	«	»
"C0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
"D0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
"E0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
"F0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

отсутствующую в вашей установке LaTeX'а; тогда этот рецепт не сработает.

- Семейство задается с помощью команды `\fontfamily`, единственный аргумент которой — внутреннее имя семейства. Пример: `\fontfamily{cmr}` обозначает семейство Computer Modern Roman.
- Насыщенность задается с помощью команды `\fontseries`, единственный аргумент которой — внутреннее обозначение для насыщенности. Пример: `\fontseries{bx}` обозначает полужирный шрифт (см. таблицу на с. 366).
- Начертание задается с помощью команды `\fontshape`, единственный аргумент которой — обозначение (опять же «внутреннее») для начертания. Пример: `\fontshape{it}` означает курсивное начертание.

Таблица B.8. Кодировка X2

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	I	⟨	⟩
"10	“	”	⌢	‶	̮	–	—		₀	`	´	-	Ԩ	ԩ	δ	δ
"20	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	Æ	Ђ	Ћ	Ҽ	Ҿ	Ҡ	Ҟ	Ӆ	I	J	Љ	Ӎ	Њ	Ҩ	Ҧ
"50	Ҏ	Q	Ҭ	S	Ҵ	Ц	Ҹ	W	Ѣ	Ѫ	Ѵ	[	\	]	^	_
"60	‘	æ	ђ	ћ	ҽ	ҿ	ҡ	ҟ	ӆ	i	j	љ	ӎ	њ	ҩ	ҧ
"70	ҏ	q	ҭ	s	ҵ	ц	ҹ	w	ѣ	ѫ	ѵ	{	|	}	˜	-
"80	Ґ	Ғ	Ӷ	Ҕ	Һ	Җ	Ҙ	Ӡ	Ї	Қ	Ҝ	Ӄ	Ԓ	Ң	Ҥ	Ӈ
"90	Ө	Ҫ	Ў	Ү	Ұ	Ҳ	Ӽ	Ӌ	Ҷ	Є	Ә	Ԑ	Ё	№	¤	§
"A0	ґ	ғ	ӷ	ҕ	һ	җ	ҙ	ӡ	ї	қ	ҝ	ӄ	ԓ	ң	ҥ	ӈ
"B0	ө	ҫ	ў	ү	ұ	ҳ	ӽ	ӌ	ҷ	є	ә	ԑ	ё	„	«	»
"C0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
"D0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
"E0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
"F0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

- Размер задается с помошью команды `\fontsize`, имеющей *два* аргумента: первый из них указывает кегль, а второй — интерлиньяж (в пунктах). Пример: `\fontsize{10}{12}`. Интерлиньяж, указанный во втором аргументе команды `\fontsize`, будет умножен на коэффициент, являющийся значением команды `\baselinestretch` (см. разд. III.9.5) (по умолчанию, как мы помним, этот коэффициент равен единице). Теперь становится ясно, как работает прием для изменения интерлиньяжа во фрагменте текста, описанный на с. 148.

При этом само по себе задание атрибутов с помощью этих команд шрифта еще не меняет: чтобы шрифт действительно сменился, надо сначала задать одну или несколько приведенных выше команд, меняющих атрибуты, но непосредственно после них необходимо дать еще команду `\selectfont`.

Через эти команды выражаются и команды более высокого уровня (скажем, \textbf или \emph). Например, \normalfont в стандартных классах задается по существу так:

```
\newcommand*{\normalfont}{%
\fontencoding{OT1}\fontfamily{cmr}%
\fontseries{m}\fontshape{n}%
\selectfont
}
```

Остается объяснить, как проинформировать LaTeX, какие шрифты соответствуют тем или иным комбинациям атрибутов. Это делается с помощью специального файла с расширением fd. Вот, например, файл, который (в использованной нами русификации) подключает шрифты типа гельветика.

```
\ProvidesFile{ot1zhr.fd}
  [Quasi Helvetica font definitions]
\DeclareFontFamily{OT1}{zhr}{}
\DeclareFontShape{OT1}{zhr}{m}{n}
   {  <5> <6> <7> <8> zhr5
      <9> <10> <10.95> zhr10
      <12> <14.4> zhr12
      <17.28><20.74><24.88> zhr12}{}
\DeclareFontShape{OT1}{zhr}{b}{n}
   {  <5> <6> <7> <8> <9> <10> <12>
      <10.95> <14.4> <17.28> <20.74> <24.88>
       zhb10}{}
\DeclareFontShape{OT1}{zhr}{bx}{n}
   {  <5> <6> <7> <8> <9> <10> <12>
      <10.95> <14.4> <17.28> <20.74> <24.88>
      zhbx10}{}
\DeclareFontShape{OT1}{zhr}{m}{sl}
    { <5> <6> <7> <8> <9> <10> <12>
      <10.95> <14.4> <17.28> <20.74> <24.88>
      zhsl10}{}
\DeclareFontShape{OT1}{zhr}{m}{it}{ <-> sub * zhr/m/sl }{}
```

Сначала говорится, что определяется семейство zhr в кодировке OT1. (Что не вполне верно, так как в OT1 никаких русских букв нет.) Затем объясняется, что́ надо использовать в качестве светлых шрифтов прямого (не курсивного) начертания: шрифты zhr5, zhr10, zhr12 (выбор зависит от размера нужного шрифта, указанного в угловых скобках). Затем объясняется, что́ надо использовать в качестве полужирных (bx) и наклонных (`sl`) шрифтов. Наконец, в последней строке говорится, что (по причине отсутствия курсива в шрифтах типа гельветика) вместо курсива (`it`) надо пользоваться наклонным (`sl`) шрифтом.

Как видите, ничего сложного в подключении нового семейства нет, если внутренняя кодировка шрифтов совпадает с кодировкой семейства, для которого они применяются. Если же нет, то нужно изготавливать виртуальные шрифты, описание чего выходит за рамки этой книги (есть по крайней мере два способа: один описан в документации к драйверу `dvips`, второй содержится в пакете `fontinst`).

В заключение еще раз скажем, что все эти (полезные скорее для общего понимания ситуации, чем практически) сведения приведены здесь в качестве затравки — чтобы вдохновить читателей на изготовление хороших и доступных для применения русских шрифтов.

Может, попробуете?

Приложение Г

Классы документов AMS

Американское математическое общество (AMS) распространяет три специализированные класса документов, предназначенные для набора математических текстов: `amsart`, `amsproc` и `amsbook`. Оставляя последний в стороне (уж если Американское математическое общество закажет вам монографию, то, наверное, снабдит и подробными инструкциями), остановимся на особенностях оформления документа «в целом», характерных для первых двух классов.

Для начала отметим, что AMS'овские классы документов автоматически подключают стилевые пакеты `amsmath` и `amsthm` (если, однако, вам понадобились коммутативные диаграммы или дополнительные шрифты, то пакеты `amscd`, `amsfonts` или `amssymb` все же придется подключить в явном виде, с помощью команды `\usepackage`). Остальные особенности этих классов относятся к титульной информации (т. е. к тому, что идет до команды `\maketitle`) и рубрикации документа.

Начнем с титульной информации. Команды `\title` и `\author` могут принимать необязательный аргумент (в квадратных скобках, естественно), ставящийся перед обязательным. Эти необязательные аргументы суть сокращенные варианты заглавия и имени автора, предназначенные для включения в колонтитулы. Команда `\thanks`, в аргументе которой обычно выражается благодарность за финансовую поддержку, оформляется не как сноска к имени автора, а записывается в преамбулу самостоятельно, наравне с `\author` и `\title`. В преамбуле одного документа может быть несколько команд `\thanks`. Команда `\address` также принимает один обязательный аргумент — адрес автора (можно разбивать на строки командой `\\`). Если вы хотите, наряду с постоянным адресом, указать еще и адрес для текущей переписки, можно это сделать в аргументе команды `\curraddr`. А если автор хочет указать и электронный адрес, можно его указать в команде `\email` (вниманию привыкших к $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX'у: символ `@` надо набирать непосредственно, не используя backslash и/или удвоения!). Команда `\address` должна следовать *после* команды `\author`; если вы пользуетесь командами `\curraddr` и/или `\email`, то они, в свою очередь, должны следовать после команды `\address` именно в таком порядке, как указано выше.

Все сказанное относилось к случаю, когда автор у документа один. Если авторов несколько, то информацию о каждом из них надо задавать *отдельной* командой `\author`; после каждой команды `\author` ставится своя команда `\address`, а также, если нужно, `\curraddr` и/или `\email`.

Если вы хотите посвятить кому-то свою работу, запишите это посвящение в аргументе команды `\dedicatory`. В аргументе команды `\keywords` вы можете записать список ключевых слов, определяющих вкратце, что надо знать для понимания вашего труда, а в аргументе команды `\subjclass` — указать, к какому разделу математики, согласно рубрикатору AMS, относится ваш опус. Наконец, аннотация к статье, оформленная как окружение `abstract`, в AMS'овских классах должна идти *до* `\maketitle` (но все равно после `\begin{document}`).

Что касается самого текста статьи, то главное отличие от того, что нам привычно по «обычному» LaTeX'у, состоит в командах рубрикации. Классы `amsart` и `amsproc` допускают в качестве таковых только `\section`, `\subsection`, `\subsubsection`, а также команду `\specialsection`, задающую самые крупные рубрики (более крупные, чем `\section`).

Теперь вы знаете достаточно, чтобы оформить статью по математике в соответствии с канонами Американского математического общества. Осталось только эту статью написать. Успехов вам!

Приложение Д

Пакет XY-pic

Как мы отмечали в разд. II.4.3, возможностей пакета `amscd` для печати «коммутативных диаграмм» хватает не всегда: бывают нужны стрелки наклонные, изогнутые и т. д. Для печати таких диаграмм можно воспользоваться стилевым пакетом XY-pic.

Этот пакет — большой и сложный, и в настоящем приложении мы рассмотрим только те его возможности, что относятся непосредственно к коммутативным диаграммам (обо всем прочем можно почитать в оригинальной документации, распространяющейся в электронном виде вместе с пакетом).

1. Пример с комментариями

Чтобы можно было пользоваться пакетом XY-pic для набора коммутативных диаграмм, надо подключить стилевой пакет `xy` с опциями `matrix`, `arrow` и `curve`. Иными словами, надо написать в преамбуле

```
\usepackage[matrix,arrow,curve]{xy}
```

Объяснить, как набирать с помощью XY-pic'а коммутативные диаграммы, удобно на примере.

Рассмотрим следующую диаграмму:

$$\begin{tikzcd}
 & & M' \arrow[dl, "e"] \arrow[ddll, dashed, bend right, "u"'] & & \\
 & K \arrow[rr, "f"] \arrow[dr, "h"] & & L \arrow[ul, "a"'] \arrow[dl, "g"'] & \\
L' \arrow[ur, "d"'] \arrow[rrrr, dashed, bend right, "v"'] & & M \arrow[ll, "c"'] \arrow[rr, "p"] & & K' \arrow[ul, "b"']
\end{tikzcd}$$

Ей соответствовал такой исходный текст:

```
$$
  \xymatrix{
  && M'\ar@{o->}[dl]^e \ar@/_1pc/@{-->}[ddll]_u\\
  & K\ar[rr]^f \ar[dr]^h && L \ar[ul]_a \ar[dl]_g\\
  L'\ar@{o->}[ur]_d \ar@/_1pc/@{-->}[rrrr]_v &&
      M\ar[rr]^p \ar[ll]_c && K'\ar@{o->}[ul]_b
  }
$$
```

Разберем его шаг за шагом.

Вся диаграмма записывается в аргументе команды `\xymatrix`.

Далее, диаграмма состоит из формул, соединенных стрелками. Прежде чем набирать исходный текст для диаграммы, надо мысленно расположить эти формулы в вершинах прямоугольной решетки. В нашем случае это решетка 5×3: M' стоит в третьей позиции верхней строки, K и L — во второй и четвертой позициях средней строки, L', M и K' — в первой, третьей и пятой позициях нижней строки. Строки разделяются командами `\\`, элементы строки — символами `&`. Если в каких-то узлах решетки ни одной формулы нет, надо оставить пустое место; символы `&` в необходимом количестве присутствовать обязаны.

После каждой из формул (и до следующего `&` или `\\`) следуют обозначения для всех стрелок, *выходящих* из этой формулы. Рассмотрим эти обозначения повнимательнее.

Каждое обозначение для стрелки состоит из пяти элементов (не все эти элементы обязательны).

Первый (обязательный) элемент обозначения для стрелки — команда `\ar`.

Второй элемент — обозначение для изгиба стрелки (если стрелка прямая, его можно опустить). Оно имеет вид `@/.../`, где на месте точек записывается указание о том, как именно эту стрелку надо изогнуть. Это указание состоит из символа `^` или `_`, за которым следует длина в TeX'овских единицах, указывающая степень изогнутости (в нашем примере у всех изогнутых стрелок эта длина равна `1pc`). Длину можно и не указывать, написав просто `@/^/` или `@/_/`, — тогда стрелке буден придан некоторый изгиб «по умолчанию». В любом случае символ `^` или `_` указывает, в какую сторону стрелка изгибается: если `_`, то вправо, если `^`, то влево (если смотреть от начала стрелки к ее концу).

Третий элемент — указание на начертание стрелки (если стрелка «обычная», его можно опустить). Оно имеет вид `@{...}`, где на месте точек ставится условное обозначение, более или менее имитирующее требуемую форму. В нашем примере присутствуют пунктирные стрелки, для которых это обозначение имеет вид `@{-->}`, и стрелки с кружочком в начале, обозначаемые как `@{o->}`.

Четвертый (обязательный) элемент обозначения для стрелки указывает ее направление. Каждая стрелка рассматривается как идущая из одного узла решетки в другой. Для задания направления (или, если угодно, точки назначения) стрелки необходимо поместить в квадратные скобки комбинацию из букв `u` (вверх), `d` (вниз), `r` (вправо) и `l` (влево). Например, `[ddll]` означает, что пункт назначения стрелки находится на нашей решетке на два шага вниз и на два шага влево от той формулы, из которой стрелка выходит.

Пятый и последний элемент (необязательный) определяет надпись при стрелке. Он состоит из символа `^` или `_` и текста надписи (если в надписи больше одного символа, ее надо, как водится, взять в фигурные скобки). Знак `^` указывает, что надпись должна быть слева от стрелки (если смотреть от начала к концу), знак `_` — что справа. При одной стрелке могут быть надписи с обеих сторон.

2. Некоторые общие правила

Синтаксис XY-pic'а весьма сложен, и мы не будем пытаться изложить его полностью (автор должен признаться, что в полном объеме он его и не знает). Вместо этого приведем несколько приемов, наиболее, на наш взгляд, полезных на практике.

2.1. Управление расположением надписей

По умолчанию надпись при стрелке, соединяющей две формулы, располагается посередине между центрами этих формул. Если размеры этих двух формул различны, надпись при этом оказывается слишком близко к одной из них, что нехорошо. В таком случае можно дать указание, чтобы надпись располагалась в середине именно стрелки: поставить между `^` или `_` и надписью знак `-` (минус):

$$\begin{tikzcd} A\times B\times C \arrow[dr, dotted, "f_1"] & \\ & D \end{tikzcd}$$

```
$\xymatrix{
A\times B\times C\ar@{.>}[dr]^-{f_1}\\
&D
}$
```

Заодно мы продемонстрировали еще одно возможное начертание стрелки.

Можно также в явном виде указать, в каком месте между центрами формул, соединяемых стрелками, надо сделать надпись. Для этого надо между `^` или `_` и надписью поставить в круглых скобках десятичную дробь из интервала $(0;1)$ (скажем, `(0.25)` означает, что надпись должна быть на четверти пути из начала в конец):

$$A \xrightarrow{f} B$$

```
$\xymatrix@1{
A\ar[rr]^(.25){f}&&B
}$
```

Заодно мы применили констукцию, рекомендуемую авторами пакета для ситуаций, когда вся «диаграмма» укладывается в одну строчку: поместили `@1` между `\xymatrix` и открывающей фигурной скобкой; утверждается, что в этом случае диаграмма будет выглядеть более удачно.

Наконец, можно сделать так, чтобы надпись была не сбоку от стрелки, а разрывала стрелку; для этого надо вместо `^` или `_` написать `|`, как в следующем примере:

$$\begin{tikzcd} A \arrow[dr, bend left, "f" description] & \\ & B \end{tikzcd}$$

```
$\xymatrix{
A\ar@/^/[rd]|{f}\\
&B
}$
```

2.2. Сдвинутые стрелки

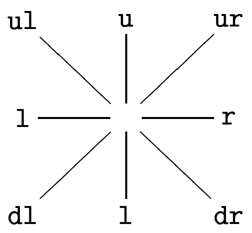
Стрелки можно сдвигать параллельно себе. Для этого используется конструкция `@<...>`, где на месте точек стоит длина в TeX'овских единицах, указывающая величину сдвига. Если эта длина положительна, то сдвиг будет влево (если смотреть от начала к концу стрелки), если отрицательна, то вправо:

$$\begin{tikzcd} A \arrow[dr, shift left, "f"] \arrow[dr, shift right, "g"'] & \\ & B \end{tikzcd}$$

```
$\xymatrix{
A\ar@<1ex>[dr]^f
\ar@<-1ex>[dr]_g\\
&B
}$
```

2.3. Еще об изогнутых стрелках

Кроме конструкции `@/.../`, изгиб стрелки можно задавать конструкцией `@(`*направление_выхода, направление_входа*`)`, где *направление_выхода* и *направление_входа* — это буквы или пары букв, значение которых показано на следующем рисунке:



Пример:

$f \circlearrowleft A \xrightarrow{g} B$

```
$\xymatrix{
A\ar@(ul,dl)[]_f\ar[r]^g&B
}$
```

Заодно показано, как с помощью «пустого» указателя направления `[]` и `@(...)`-конструкции можно напечатать стрелку, ведущую из формулы в нее же.

2.4. Начертания стрелок

В следующей таблице собраны некоторые начертания стрелок (часть из них стрелками, строго говоря, не являются, но в диаграмме могут быть полезны), которые можно получить с помощью `@{...}`-конструкции.

$\dashrightarrow$	`@{-->}`	$\Longrightarrow$	`@{=>}`	$\cdots\!\!\!\to$	`@{.>}`
$\rightarrowtail$	`@{>->}`	$\Rrightarrow$	`@3{->}`	$\longleftrightarrow$	`@{<->}`
$\twoheadrightarrow$	`@{->>}`	$\Longleftrightarrow$	`@{<=>}`	$\longmapsto$	`@{|->}`
$\hookrightarrow$	`@{^{(}->}`	$\rightharpoonup$	`@{-^{>}}`	$\rightharpoondown$	`@{-_{>}}`
——	`@{-}`	$=$	`@{=}`	$\equiv$	`@3{-}`

Если того, что приведено в таблице, недостаточно, читатель может попробовать скомпоновать еще что-нибудь в этом роде по аналогии (вероятность, что это сработает, отлична от нуля) или обратиться к документации.

2.5. Оптимизация и предупреждение ошибок

Пакет XY-pic заставляет TEX работать буквально на пределе возможностей; файлы, в которых используются XY-pic'овские конструкции, обрабатываются довольно медленно. Для ускорения работы полезно включить в преамбулу команду `\CompileMatrices`: в этом случае при первом запуске LATEX'а информация о ваших `\xymatrix` будет записана в специальные файлы, а при последующих запусках обрабатываться будут именно они, что сэкономит TEX'у время на развертывание (части из) чудовищного количества макроопределений.

Иногда синтаксис XY-pic'а вступает в конфликт с синтаксисом других LATEX'овских команд, что приводит к весьма загадочным сообщениям об ошибках. Чтобы избежать этого, при пользовании XY-pic'ом полезно применять следующие меры предосторожности:

- в аргументе команды `\xymatrix` каждую из формул, соединяемых стрелками, и каждую надпись при стрелке стоит заключать в фигурные скобки (автор поленился это сделать в вышеприведенных примерах, но в них формулы были очень просты, что снижает шанс нарваться на неприятность);

- если в вашей формуле присутствует что-то еще, кроме одной-единственной команды `\xymatrix`, возьмите, от греха подальше, всю эту команду в фигурные скобки, вот так:

  ```
  {\xymatrix{....}}
  ```

- не пытайтесь определять собственные сокращенные обозначения для того, что может быть в аргументе команды `\xymatrix`: по TEX'ническим причинам эти макросы могут не сработать.

Еще одно замечание: пакет XY-pic (как и подавляющее большинство LATEX'овских стилевых пакетов) распространяется бесплатно; тем не менее, если вы пользуетесь им в своем документе, то авторы этого пакета очень просят вас отразить это обстоятельство в сноске к статье или в выходных данных книги.

Будем надеяться, что описанные выше возможности XY-pic'а достаточны для набора ваших диаграмм. Как мы уже отмечали, этот пакет предоставляет гораздо бóльшие графические возможности, но если нужна сложная графика, разумнее освоить программу `metapost`, подготавливающую рисунки в формате PostScript, и включать в текст полученные с ее помощью графические файлы так, как описано в приложении Б.

Приложение E

LaTeX в России

1. Правда жизни

В основном тексте книги не раз говорилось, что LaTeX позволяет автору подготовить текст книги на домашнем компьютере, а затем послать его коллегам или в издательство, где его напечатают без проблем и в том самом виде, как автор хочет.

Настало время признаться, что сказанное несколько идеализирует реальность, даже если речь идет об английских текстах, и грубо приукрашивает ее, если речь идет о русских.

Прежде всего, не все издательства вообще знают про TeX. В этом случае автор должен принести готовый оригинал-макет в каком-нибудь известном издательству формате. Чаще всего используется формат PostScript. Теоретически такой файл достаточно передать побайтно в фотонаборное устройство или PostScript-принтер, и текст будет напечатан в нужном виде. На практике, однако, возможны осложнения. В одном популярном московском издательстве не только не умели работать с TeX-файлами, но и PostScript-файл умудрились напечатать с непреднамеренным изменением масштаба, причем разным для разных страниц! (Что не так удивительно, если учесть, что в современных системах с дружественным интерфейсом не всегда легко предвидеть последствия того или иного движения мыши.)

«Патриотически» настроенным читателям будет приятно узнать, что эти проблемы отнюдь не ограничиваются Россией: в одном широко известном американском издательстве на компьютере было установлено несколько версий LaTeX'а, причем сотрудники издательства этого не осознавали и удивлялись, почему обработка одних и тех же файлов иногда дает слегка разные результаты. Проблема тут в том, что время от времени в LaTeX и стилевые пакеты к нему вносятся улучшения (исправляются старые ошибки и вносятся новые).

Вот еще одна, совсем свежая, история. В первоначальной версии этой книги объяснялось, что для двойных акцентов в математических формулах (типа $\hat{\hat{a}}$) надо использовать команду `\Hat`, а не `\hat`, поскольку команда `\hat` ставит акценты со сдвигом (см. с. 72), и был пример плохого двойного акцента. При обработке текста книги с новой версией пакета `amsmath` выяснилось, что авторы этого пакета о нас позаботились и изменили команду `\hat`, из-за чего пример ее ошибочной работы пришлось создавать искусственно.

Впрочем, надо отдать должное разработчикам LaTeX'а и макропакетов к нему: по сравнению с другими «программными продуктами» количество версий (LaTeX обновляется раз в год, а раньше — раз в полгода) невелико, а различия между ними незначительны. (Plain TeX меняется еще реже и меньше.)

2. Русскоязычные беды

С русскими текстами дела обстоят еще хуже. Если вы подготовите файл с русским текстом и пошлете его своему коллеге, у которого используется другая операционная система или просто другая русификация LaTeX'а, почти наверняка у него возникнут проблемы и текст не напечатается или напечатается не так.

Прежде чем описывать это бедственное положение более подробно, попытаемся хотя бы немного оправдаться и объяснить, отчего так получается.

Естественно, что Дональд Кнут прежде всего заботился об англоязычных авторах. Если мы хотим использовать TeX для текстов на других языках, возникает несколько проблем.

- Нужны буквы, которых нет в английском языке. Например, в немецком языке нужны буквы с умлаутами (типа ä), а в скандинавских языках используется буква ø — и мудрый Кнут заранее предусмотрел ее в разработанных им шрифтах (буквы же с умлаутами можно собирать из обычных букв и диакритических знаков). Хуже обстоит дело с русским (а также, например, греческим) языком, где нужны не отдельные новые буквы, а целый алфавит.

- Даже если нужные буквы есть, возникает проблема переносов. В разных языках правила переноса различны, и потому таблица переносов для английского языка будет давать неверные результаты при обработке (скажем) немецких текстов.

- Наконец, в разных странах разные типографские традиции, и потому сверстанный по англо-американским канонам текст на другом языке кажется странным. (В частности, названия разделов документа для разных языков разные — но есть и гораздо более тонкие различия.)

Эти проблемы решались постепенно. Начальный вариант TEX'а использовал шрифты из 128 символов и предусматривал только английские переносы. Затем Кнут добавил возможность хранить таблицы переноса для нескольких языков. Впрочем, это не решало проблем с переносом иноязычных слов, поскольку буквы с диакритическими знаками не были полноценными буквами с точки зрения алгоритмов переноса. К счастью, байт оказался восьмибитовым, и поэтому TEXнически несложно было перейти к шрифтам из 256 символов. В эти 256 символов уже помещались буквы большинства языков на латинской графической основе[1], и для LATEX'а была разработана соответствующая кодировка (T1), в которой уже все буквы полноценные. (Не торопитесь, впрочем, завидовать европейцам: не успели они порадоваться достигнутой стандартизации, как оказалось, что символ для европейской денежной единицы вставить в T1 некуда, и он, в отличие от более удачливого доллара, был добавлен в кодировку TS1).

Параллельно TEX приспосабливался к русскому языку. Прежде всего нужно было нарисовать русские буквы, которые были бы по внешнему виду аналогичны буквам из кнутовских шрифтов. Это было сделано, причем несколькими способами (см. приложение В, разд. 5).

Тем не менее мечты о стандартизации («Чтобы использовать русские буквы, подключите такой-то стандартный стилевой пакет») пока остаются мечтами. Помимо субъективных причин (все привыкли к своим любимым вариантам русификации и не хотят переучиваться), есть и объективные проблемы.

Первая связана с невозможностью соединить в одном шрифте все желательные символы, поскольку шрифты более чем с 256 символами TEX (без серьезных переделок) не использует, а свободных мест в кодировке T1 нет. Более того, даже если отказаться от символов европейских языков, но хотеть разместить все известные буквы кириллического типа, то и тогда оказывается недостаточным 256 символов. Поэтому есть несколько стандартизованных кодировок, включающих русские и другие кириллические буквы (T2A, T2B,...), не говоря уже о других, не стандартизованных, вариантах.

Вторая связана с тем, что существует несколько способов представления русских букв в файлах. Наиболее распространены три: koi8-r (обычно она используется в UNIX-подобных системах), cp866 (DOS)

[1]Хотя и не всех: например, некоторых букв, использующихся в латышском или литовском языках, в кодировке T1 нет.

и cp1251 (Windows). Поэтому само понятие совместимости становится спорным: если DOS-пользователь приносит своему Windows-собрату дискету с файлом в кодировке cp866, должен ли этот файл обрабатываться LaTeX'ом в системе Windows без дополнительного перекодирования или после перекодирования в cp1251? Должна ли одна и та же программа обрабатывать тексты в разных кодировках (а может, даже и тексты, разные части которых записаны в разных кодировках)? Этого можно достичь, применяя dirty tricks (сделав русские буквы «активными символами» — читатели книги [2] нас поймут) — но тогда нельзя использовать русские буквы в именах команд. И т. д. и т. п.

3. Что делать?

Но все-таки, что же делать бедному автору, если ему нужно использовать LaTeX для обработки текстов не на английском языке? Главный совет — спросить у знакомых, чем они пользуются, и попытаться воспользоваться теми же самыми средствами. Для языков на латинской основе может пригодиться пакет `babel` — ниже мы скажем о нем несколько слов в разд. Е.6. С русским дело сложнее — даже при использовании пакета `babel` могут потребоваться действия, выходящие за рамки обычного подключения пакета, поскольку во многих распространенных комплектах (teTeX, MiKTeX и др.) в готовых форматах не предусмотрена обработка русских переносов.

В предисловии к русскому переводу книги [6] приводится список из 12 различных вариантов русификации LaTeX'а (и это не полный список, как сказано в том же предисловии). Нет никакой возможности описать их все, к тому же с большинством из них автор и не знаком. Вместо этого мы в трех следующих разделах кратко опишем следующие три вещи: вариант русификации, который использовался при подготовке этой книги (его можно найти на приложенном к книге диске), русификацию LaTeX'а с помощью пакета `babel`, и использование того же пакета `babel` для подготовки текстов на западноевропейских языках, отличных от английского.

4. Пример: использованная в книге русификация

Для обработки русских текстов вместо стандартной команды `latex` используется специальная команда `rlatex`. При этом тексты записываются в кодировке cp866 для DOS и Windows и в кодировке koi8-r для UNIX-подобных систем. Подключение дополнительных пакетов не обязательно: и без них русские буквы можно использовать наравне с латинскими (в том числе в названиях команд, счетчиков, меток и др.), переносы учитывают таблицу русских переносов, можно использовать команды `\лк`, `\пк` для кавычек и `\номер` для знака номера. Буква ё является одной из 33 равноправных букв русского алфавита.

Само по себе использование команды `rlatex` не меняет оформления текстов (и для английских текстов она дает почти тот же результат, что и команда `latex`). Но можно обычным способом (с помощью `\usepackage`) подключить пакет `russcorr`. Это приведет к следующим изменениям:

- по умолчанию устанавливается режим `\frenchspacing`, а значение параметра `\righthyphenmin` полагается равным 2;
- определены команды `\tg`, `\ctg`, `\cosec`, `\arctg`, `\sh`, `\ch`, `\th`, `\cth`, дающие принятые в России названия этих функций;
- становятся определены команды `\varkappa`, `\eps` и `\vphi`, дающие принятые в русских текстах начертания греческих букв $\varkappa, \varepsilon, \varphi$;
- команды `\leqslant` и `\geqslant` дают (даже без подключения пакета `amssymb`) знаки неравенств $\leqslant$ и $\geqslant$, принятые в русских текстах;
- переведены на русский заголовки «Оглавление», «Рис.», «Часть», «Список рисунков», «Список таблиц», «Литература», «Предметный указатель», «Таблица», «Аннотация», «Глава», «Приложение», «Доказательство» (последний заголовок нужен при использовании пакета `amsthm`);
- в заголовках разделов документа после номера раздела ставится точка (чего обычный LaTeX не делает); можно заменить эту точку на что-то другое, переопределив команды `\postchapter`, `\postsection` и т. п.; кроме того, имеются команды `\presection`, `\presubsection` и т. д. которые изначально ничего не добавляют перед номером раздела (подраздела и т. д.), но также могут быть переопределены (например, можно получить знак параграфа перед номерами разделов, если переопределить `\presection` как `\S`);
- точки ставятся также после названий рисунков и таблиц (вместо двоеточий) и после номеров теорем;
- имеются команды `\ralph` и `\Ralph`, являющиеся русскими аналогами команд `\alph` и `\Alph`; при этом буквы `ё`, `й`, `ъ`, `ы` и `ь` пропускаются;

- оформление перечней в окружении `enumerate` меняется: элементы верхнего уровня нумеруются цифрами со скобками, а не с точками, на втором уровне используются русские буквы вместо латинских, на третьем уровне элементы вообще не нумеруются, а отмечаются тире; четыре уровня вложенности не разрешаются;
- добавляется окружение `rlist`, которое нумерует пункты русскими буквами по алфавиту (строчными; чтобы получить прописные, надо использовать необязательный аргумент `[u]`);
- приложения (при использовании команды `\appendix`) нумеруются прописными русскими буквами вместо латинских;
- изменено поведение команды `\cleardoublepage`: оставляемая ей пустая четная страница теперь совсем пуста (не содержит колонтитулов, номера страницы и т. п.).
- команды `\glqq` и `\grqq` задают кавычки-„лапки“;
- при использовании стилевой опции `longtoc` в оглавление помещаются не сокращенные, а полные названия разделов;
- при использовании стилевой опции `indentheadings` первые абзацы и заголовки разделов печатаются с абзацным отступом;
- При использовании стилевой опции `yo` слова с буквой ё получают возможность нормально переноситься, даже если в исходном тексте эта буква задана как `\"e`.

Наконец, в состав русифицирующего комплекта входит специальный вариант программы `makeindex`, используемой при подготовке предметного указателя; этот вариант учитывает порядок русских букв при сортировке.

Все необходимые файлы для установки указанного варианта русификации (дополнительные файлы для UNIX, комплект TeX (конкретно — MiKTeX) плюс GhostScript и GSView для Windows, комплект emTeX для DOS) имеются на прилагаемом к книге компакт-диске (программы GSView там может не быть по лицензионным причинам, и её придётся списывать из Интернета, см. с. 415), а также доступны по `ftp`. См. приложение Ж.

5. Пример: babel для русского языка

В некоторые установки системы TeX (в частности, в некоторых русских версиях Linux) включены средства для обработки русских текстов с помощью пакета `babel`. (Говоря TEXнически, нужно, чтобы (1) сам пакет был установлен в вашей TEX-системе и (2) таблица переносов для русского языка была включена в формат.) Чтобы выяснить, так ли это, создайте файл `tstbabel.tex` следующего содержания:

```
\documentclass{article}
\usepackage[russian]{babel}
\begin{document}
Test text.
\end{document}
```

и обработайте его командой `latex tstbabel.tex`. Если обработка файла прошла успешно (TEX не пожаловался на отсутствие пакета `babel` или неизвестную стилевую опцию `russian`), это значит, что пакет `babel` с русскими дополнениями установлен. Теперь надо проверить наличие русской таблицы переносов; для этого загляните в получившийся файл `tstbabel.log`. Его начало должно выглядеть примерно так:

```
This is TeX, Version 3.14159 (Web2C 7.3.1)
**tstbabel.tex
(tstbabel.tex
LaTeX2e <2001/06/01>
Babel <v3.7h> and hyphenation patterns for english, french,
german, ngerman, russian, nohyphenation, loaded.
```

Если в списке языков (он идет после слов «`hyphenation patterns for`») указан русский (`russian`), то таблица переносов для русского языка включена в формат и можно набирать русские тексты с помощью `babel`. (Если она в формат не включена, то можно попробовать это исправить. Необходимые действия зависит от версии используемой TEX-системы; для этого даже есть специальные «утилиты», например, программа `MiKTEX Options` в системе MiKTEX или программа `texconfig` в системе teTEX.)

Чтобы использовать `babel` с русскими текстами, необходимо включить в преамбулу строки

```
\usepackage[...]{inputenc}
\usepackage[russian]{babel}
```

В квадратных скобках вместо `...` нужно указать кодировку, в которой будет набираться русский текст: например, `cp866` (DOS-кодировка), `cp1251` (Windows-кодировка) или `koi8-r` (кодировка koi8-r). Эти строки должны находиться в преамбуле *до* вызова каких-либо пакетов, использующих русские буквы. После этого в тексте можно набирать русские буквы в указанной кодировке. Важное ограничение: русские буквы нельзя использовать в именах команд или меток!

Пакет babel с опцией russian определяет множество командных последовательностей и лигатур для набора различных символов: например, с помощью "</"> или <</>> можно получить кавычки-«елочки», "--- дает тире. Кроме того, переопределяются команды для набора заголовков, списка литературы и проч. (например, команда \chapter теперь будет печатать слово «Глава», а не «Chapter» — ср. предыдущий раздел). Более подробную информацию можно найти в документации.

Использование пакета babel несовместимо с описанной выше (и использованной в книге) русификацией LaTeX'а, а также с некоторыми другими пакетами — не пытайтесь их соединить! Кроме того, вам могут потребоваться некоторые специальные средства для подготовки предметных указателей, чтения сообщений об ошибках и др.

6. Пример: babel для французского языка

Предположим, что вам надо подготовить с помощью LaTeX'а текст на французском языке. Тогда файл надо начать так:

```
\documentclass[12pt,a4paper]{article}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
⟨другие команды \usepackage, если нужно⟩
```

После этого в тексте будут при необходимости делаться переносы по французским правилам, в том числе и в словах, включающих буквы с диакритическими знаками («аксанами», как сказал бы любитель французского), невзирая на то, что эти буквы задаются в тексте с помощью команд \', \^ и т. п. Стандартные заголовки («глава», «список литературы» и т. п.) и дата (проставляемая автоматически или с помощью команды \today) будут печататься по-французски. Более того, в тексте будут автоматически учитываться различные французские полиграфические тонкости (например, некоторые знаки препинания будут печататься не вплотную к слову, а с небольшим отступом, как это во Франции и принято). Кавычки-«елочки», принятые во французских текстах, можно при этом набирать как << и >> (в шрифтах, используемых пакетом babel, это лигатуры). Для некоторых других типично французских полиграфических эффектов предусмотрены специальные команды, с которыми можно ознакомиться в документации к пакету.

Кроме французского, аналогичным образом можно обойтись с немецким (вместо french надо написать german), итальянским и многими другими европейскими языками; впрочем, надо иметь в виду, что далеко не для всех из них будут доступны автоматические переносы (см. предыдущий раздел по поводу того, как выяснить, включена ли в формат нужная вам таблица переносов и как можно попытаться ее добавить). Если вы не уверены, что таблица переносов есть, то лучше вообще запретить переносы, сказав в преамбуле `\hyphenpenalty=10000` (см. с. 127): худшее, что может случиться при использования русского `babel`'а без русских переносов, — это то, что в русских словах не будет делаться переносов, а вот при обработке, скажем, французского текста без французской таблицы переносов французские слова переноситься будут, но по английским правилам, что, надо полагать, хуже.

Приложение Ж

Откуда взять TeX?

До сих пор мы старательно уходили от разговора о том, как на практике воспользоваться системой LaTeX, — прежде всего потому, что это зависит от того, какая операционная система установлена на вашем компьютере и какая версия TeX'а выбрана. В этом приложении мы попытаемся сказать что-то более конкретное, имея в виду пользователей разных операционных систем.

1. UNIX-подобные системы

К их числу относятся популярная ныне система Linux, другая популярная система FreeBSD (обе они являются свободно распространяемыми), система Solaris фирмы Sun Microsystems и многие другие.

1.1. Как проверить наличие LaTeX'а

Скорее всего, TeX уже входит в состав имеющегося у вас комплекта. Обычно это система teTeX, и в дальнейшем мы описываем именно ее (для системы Linux, в других системах детали могут отличаться). Чтобы убедиться в том, что TeX установлен, запустите команду `tex`: должно появиться что-то вроде

```
This is TeX, Version 3.14159 (Web2C 7.3.1)
**
```

То же самое должно появляться при запуске команды `latex`. (Прервать выполнение команды `tex` обычно можно при помощи одной из клавиш Control-C и Control-D; если появляется вопросительный знак, то можно нажать клавишу `x` и затем `Enter`.)

Если этого не происходит, скорее всего, это означает, что при установке системы (из «дистрибутива») вы не заказали установку системы

TEX. В этом случае можно попытаться повторить инсталляцию, выбрав соответствующие пакеты[1], или добавить эти пакеты в систему. Например, в дистрибутиве системы Linux фирмы RedHat для этого применяется программа `rpm`. Заново устанавливать TEX, минуя принятую систему инсталляции, мы не советуем. Лучше попытаться разыскать и установить отдельные недостающие пакеты, следуя инструкциям к использованному вами дистрибутиву.

Не всегда по умолчанию устанавливается комплект TEX-документации. Советуем вам проверить и при необходимости доустановить его (скорее всего, он есть на дисках, с которых вы устанавливали операционную систему). Искать документацию следует прежде всего в директории типа `/usr/share/texmf/doc` (в ней должно быть много поддиректорий, относящихся к разным частям системы: `amstex`, `bibtex`, `context` и т. д.)

Кроме того, для работы с LATEX'ом вам понадобится X Window System (графическая подсистема UNIX, которая рисует на экране окна, курсор от мыши и т. п.). Скорее всего, она тоже уже установлена, а если нет, попросите знатоков это сделать. Если она есть, можно попробовать полный цикл обработки файла в LATEX'е.

1.2. Создание файла

Прежде всего надо создать файл (скажем, `test.tex`) такого содержания:

```
\documentclass{article}
\begin{document}
This is a test file.
\end{document}
```

Обратите внимание, что буквы здесь только латинские. Это существенно: сначала мы хотим проверить работу исходного LATEX'а, а не его русской версии. Такой файл надо создавать с помощью текстового редактора. Этих редакторов много (`emacs`, `pico`, `joe`, `jed`, старинный редактор `vi` и многие другие). Есть также программа Midnight Commander (`mc`), которая напоминает знакомую многим программу Norton Commander или Volkov Commander; в неё встроен текстовый редактор (вызывается клавишей F4). Так или иначе, если вы вообще работаете с UNIX, то скорее всего каким-то текстовым редактором пользоваться уже умеете. На всякий случай содержимое файла можно проверить командой `less test.tex` (ее работу можно завершить клавишей `q`).

[1]Здесь под «пакетами» имеются в виду не пакеты макроопределений, а «packages» в смысле инсталлятора типа RedHat или «ports» в смысле FreeBSD.

1.3. Обработка файла с помощью LaTeX'а

После того, как файл `test.tex` создан, надо дать команду

```
latex test.tex
```

На экране должно появиться примерно следующее:

```
This is TeX, Version 3.141519 (Web2C 7.3.1)
(test.tex
LaTeX2e <2000/06/01>
(/usr/share/texmf/tex/latex/base/article.cls
Document Class: article 2000/05/19 v 1.4b
Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/size10.clo))
No file test.aux
[1] (test.aux) )
Output written on test.dvi (1 page, 236 bytes).
Transcript written on test.log
```

(Такое сообщение появляется при первом запуске; при втором файл `test.aux` уже есть и на его отсутствие не жалуются.) При этом должны образоваться (помимо `test.aux`) еще файлы `test.dvi` и `test.log`. Последний файл должен содержать примерно то же самое, что было выведено на экран.

1.4. Просмотр dvi-файла на экране

Описанные действия можно было произвести и без графической системы («на консоли», как иногда говорят). Но следующее действие уже требует работающей системы X. Получив доступ к командной строке (часто это можно сделать, выбрав пункт меню «New shell», или «Xterminal», или «xterm», или еще что-то в таком роде), надо запустить программу `xdvi` командой

```
xdvi test.dvi
```

При этом должно появиться новое окно, в котором показана (возможно, не полностью) страница сверстанного текста. Справа от нее, скорее всего, будут разные «кнопки», назначение которых в основном понятно из названий. Клавиши Page Up и Page Down переходят на предыдущую и следующую страницу (пока, впрочем, страница только одна, и они не действуют), стрелки позволяют перемещаться по странице (что можно делать и с помощью мыши). Из других полезных возможностей: нажав на клавишу с цифрой, а затем на `s`, можно получить на экране изображение, уменьшенное в соответствующее число раз (по сравнению с максимально возможным, когда одна точка в шрифтах соответствует одной точке экрана) — так что `1s` дает изображение максимального размера. Аналогично, набрав число и потом нажав клавишу `g`, можно перейти на страницу с заданным номером. Еще одно полезное свойство: нажатие на кнопку мыши может действовать как лупа.

Многие другие возможности можно узнать из документации (в частности, с помощью команды `man xdvi`). Выйти из программы можно, нажав клавишу `q`.

1.5. Изготовление PostScript-файлов

Теперь следует опробовать работу с PostScript-файлами. Начнем с их создания: находясь всё в той же директории, где лежат файлы `test.tex` и `test.dvi`, дайте команду

```
dvips -o test.ps test.dvi
```

Она выдаст на экран сообщение наподобие следующего:

```
This is dvips(k) 5.86 Copyright 1999 Radical Eye Software
(www.radicaleye.com)
' TeX output 2002.07.21:2137' -> test.ps
<texc.pro> [1]
```

в котором [1] выдается при обработке первой страницы (для более длинных файлов страниц будет, естественно, больше). При этом создается файл `test.ps`.

Ради любопытства можно заглянуть в него с помощью команды `less`: там написано что-то вроде

```
%!PS-Adobe-2.0
%%Creator: dvips(k) 5.86 Copyright 1999 Radical Eye Software
<...>
userdict /end-hook known{end-hook}if
%%EOF
```

(когда мы провели эксперимент, в этом файле оказалось 109 строк).

1.6. Просмотр PostScript-файла

На практике, естественно, читать PostScript-файл невооруженным глазом не надо, а надо просматривать его с помощью команды `gv` (или `ghostview`). Дайте команду

```
gv test.ps
```

и убедитесь, что на экране появляется окно, в котором изображена первая (в данном случае и единственная) страница вашего файла. В этой программе также можно использовать клавиши Page Up и Page Down (переключаться между страницами можно также с помощью мыши и списка страниц слева). Увеличение регулируется в меню сверху. Есть и много других пунктов меню, которые можно опробовать. В частности, средней кнопкой мыши можно пометить некоторые избранные страницы, а затем записать их в файл (это будет PostScript-файл только с избранными страницами) с помощью кнопок меню. Можно также и напечатать избранные страницы, но в этом случае программа запрашивает, какова команда печати, поэтому лучше опробовать печать отдельно.

1.7. Печать PostScript-файла

Печать осуществляется командой

```
lpr test.ps
```

Правда, для этого должна быть настроена система печати. В многопользовательских системах настройка принтеров — задача системного администратора, который должен сообщить пользователям, как печатать файлы. (А задача пользователей — «достать» системного администратора настолько, чтобы он наладил бесперебойную работу принтеров.) На домашнем компьютере вы сами себе системный администратор, и печать придется настраивать самому или с помощью друзей. К сожалению, это очень по-разному делается в разных версиях UNIX и даже в разных дистрибутивах. В современных дистрибутивах Linux (типа RedHat или Mandrake) при настройке печати могут помочь специальные графические оболочки (например, `printtool` в RedHat Linux).

Подчеркнем, что вовсе не обязательно иметь PostScript-принтер, чтобы печатать PostScript-файлы. На самом деле годится почти любой принтер (который работает с операционными системами, отличными от Windows; избегайте неполноценных «Windows-only» принтеров). Система печати автоматически обрабатывает ваш файл с помощью программы `Ghostscript`, так что вам (если всё настроено правильно) не о чем беспокоиться.

1.8. Дополнительная информация

Описанных команд достаточно для полного цикла работы с системой LaTeX. Но есть и другие полезные возможности.

- Программа `ps2pdf` преобразует PostScript-файлы в файлы в формате PDF. Это удобно, если ваши читатели испытывают трудности с чтением PostScript-файлов, но умеют читать PDF-файлы. Программа `ps2pdf` использует `Ghostscript` и вызывается командой типа

  ```
  ps2pdf test.ps test.pdf
  ```

- Файлы в PDF-формате (не все, но большинство) можно просматривать программой `gv` (`ghostview`). Она же позволяет сохранить (save) файл в формате PostScript. Это же можно сделать с помощью программы Acrobat Reader фирмы Adobe (которая также распространяется бесплатно, но с некоторыми ограничениями). Наконец, есть программа `pdf2ps`, которая также использует `Ghostscript` и переводит PDF-файлы в PostScript-файлы.

- Имеются программы, позволяющие вырезать отдельные страницы (`psselect`), сливать (`psmerge`), изменять масштаб и собирать по несколько страниц на одну (`psnup` и более изощренная `pstops`), переставлять страницы для брошюровки (`psbook`) и многое другое. Они распространяются свободно и обычно также входят в дистрибутив UNIX-подобной операционной системы.

  Например, команда

  ```
  pstops "2:0L@0.7(21cm,0)+1L@0.7(21cm,14.85cm)" text.ps text2.ps
  ```

  создаёт файл `text2.ps`, каждая страница которого представляет собой две уменьшенные с коэффициентом 0.7 страницы файла `text.ps` (меняя числа, можно регулировать уменьшение и сдвиг). Команда

  ```
  psselect 5-26 text.ps text2.ps
  ```

  помещает в `text2.ps` страницы 5–26 файла `text.ps` (можно оставить все страницы, начиная с пятой, если написать `5-`). Более подробные сведения об этих и других программах можно получить, сказав (например) `man pstops`.

  Учтите, что эти программы работают не для любых PostScript-файлов (но обычно работают с файлами, выдаваемыми программой `dvips`). Зато они, к слову сказать, совсем маленькие (и написаны на языке C без всяких системно-зависимых хитростей).

- Файлы, относящиеся к TEX'у (включая документацию), в основном сосредоточены в специальной директории. В системе Linux это обычно директория `/usr/share/texmf`, в системе FreeBSD это может быть директория `/usr/local/share/texmf`. Кроме того, TEX может помещать создаваемые им файлы (например, файлы шрифтов) в директорию типа `/var/lib/texmf`.

- Для проверки наличия файлов (в тех местах, где их будут искать TEX-программы) может быть полезна программа `kpsewhich`: например, в ответ на команду

  ```
  kpsewhich graphicx.sty
  ```

  может быть выдано сообщение типа

  ```
  /usr/share/texmf/tex/latex/graphics/graphicx.sty
  ```

  с полным адресом этого файла.

- Имеется программа `mpost` (иногда `mp`, но часто `mp` обозначает другую программу), полезная для изготовления PostScript-рисунков. Полное название этой программы, сообщаемое при её вызове, — `metapost`.

- Есть несколько наборов программ (`NetPBM` и `ImageMagick`, например), позволяющих преобразовывать графические файлы из одних форматов в другие. Например, чтобы преобразовать pcx-файл `picture.pcx` в формат PostScript (точнее, Encapsulated PostScript) с помощью программы `convert`, входящей в пакет `ImageMagick`, достаточно сказать

  ```
  convert picture.pcx picture.eps
  ```

  Существует также (свободно распространяемый) графический редактор `gimp`, который умеет читать и записывать файлы различных форматов.

1.9. Русификация TEX'а в UNIX

Сейчас получают всё большее распространение русифицированные варианты системы Linux и других UNIX-подобных систем. Хочется надеяться, что скоро в их состав будет входить готовая к использованию и хорошо работающая русификация TEX'а вместе с соответствующей документацией. (А если быть совсем оптимистами, то можно надеяться, что эти русификации будут совместимы друг с другом хотя бы в той же степени, что и разные версии исходного англоязычного LaTeX'а.) Может быть, когда вы читаете эту книгу, такое уже случилось (всегда хочется надеяться на лучшее), тем более что многое в этом направлении уже делается (группой CyrTUG и её последователями, российскими производителями дистрибутивов UNIX и т. п.) Тогда вам не понадобится ни этот раздел, ни прилагаемый к книге компакт-диск.

Возможно также, что вы работаете в системе коллективного пользования, где кто-то уже установил тот или иной вариант русского LaTeX'а и успешно им пользуется. И в этом случае ваша жизнь упрощается: мы советуем поговорить с местными экспертами и узнать, как именно вызывать и использовать этот вариант, а не пытаться что-то установить самим. (Тем более что попытка установить одну русификацию поверх другой может нарушить работу обеих!)

Тем не менее возможен и тот случай, когда в вашей системе есть оригинальный английский LaTeX, не тронутый попытками русификации. Именно к этому случаю относятся все дальнейшие советы. По понятным причинам описывается именно тот вариант русификации, который использовался при подготовке этой книги.

В прилагаемом к книге компакт-диске в директории `tetex.unix` содержатся все необходимые для русификации материалы, собранные в архив `texkoi.tar.gz` (его можно найти в `ftp://ftp.mccme.ru/pub/tex` рядом с текстами этой книги; он занимает около мегабайта). Спишите этот архив на жёсткий диск и разархивируйте его с помощью команды

```
tar xzvf texkoi.tar.gz
```

(которую нужно давать в той же директории, где лежит архив). Появится директория `texkoi`; в этой директории есть файл `README`, содержащий краткое описание русификации (более подробное описание имеется в файле `DETAILS`), а также файлы `INSTALL.root` и `INSTALL.user`. Первый из них описывает действия, которые должен предпринять системный администратор, чтобы установить русскую версию LaTeX'а для всех пользователей, второй — что надо делать пользователю, не имеющему прав системного администратора, чтобы установить русскую версию LaTeX'а для себя[2].

Установка русских дополнений совсем проста, если у вас установлена система Linux в версии RedHat 6.0 и выше (или другая система,

[2]Все эти файлы написаны по-русски в кодировке koi8-r (типичной для UNIX), так что прежде всего нужно будет научиться читать и редактировать такие файлы (для чего необходимо подключить шрифты системы X Window System в этой кодировке и/или консольные шрифты). В разных версиях UNIX эти шрифты устанавливаются по-разному, и описание этого процесса мы не приводим, но без русских экранных шрифтов трудно воспользоваться русским TeX'ом.

совместимая по расположению TeX-файлов) и вы можете получить полномочия системного администратора (`root`). В этом случае достаточно:

1) получить полномочия `root`'a (например, с помощью команды `su`);
2) находясь в директории `texkoi` (полученной при распаковке архива), дать команду `sh install.sh`

Важно только до этого проверить (как это сделать, см. выше), что оригинальный LaTeX уже установлен, иначе попытки русификации ничего хорошего не дадут (и сильно затруднят последующую русификацию).

Если на втором шаге возникают сообщения об ошибках, значит, в вашей версии Linux файлы расположены несколько иначе и придется разбираться подробнее. Прочтите тогда внимательно сам файл `install.sh` и комментарии к нему в файле `INSTALL.root`.

Имеется также аналогичный файл `installbsd.sh`, который может помочь при установке русификации в системе FreeBSD и аналогичных (где TeX лежит в других директориях).

Проверьте, прошла ли русификация успешно. Для этого создайте тестовый файл `testrus.tex`, в котором будут русские буквы:

```
\documentclass{article}
\begin{document}
This is a test file. Это~--- тестовый файл.
\end{document}
```

(заметим в скобках, что можно сделать это с помощью примитивного редактора `xreditor`, который входит в состав русификации и должен к этому моменту быть уже установлен; кроме него, имеются программы `reditor-dos2unix` и `reditor-unix2dos`, которые преобразуют тексты из DOS-кодировки (cp866, иногда называемой «альтернативной») в UNIX-кодировку (koi8-r) и обратно).

Затем дайте команду

```
rlatex testrus.tex
```

Она должна выдать на экране текст

```
This is TeX, Version 3.141519 (Web2C 7.3.1)
(testrus.tex
This is Russified LaTeX2e [2000/07],
based on LaTeX2e 2000/06/01
(/usr/share/texmf/tex/latex/base/article.cls
Document Class: article 2000/05/19 v 1.4b
Standard LaTeX document class
```

```
(/usr/share/texmf/tex/latex/base/size10.clo))
No file testrus.aux
(/usr/share/texmf/tex/latex/base/t1cmr.fd)[1]
(testrus.aux) )
[1] (testrus.aux) )
Output written on testrus.dvi (1 page, 284 bytes).
Transcript written on testrus.log.
```

(или аналогичный) и создать файл `testrus.dvi`. При просмотре этого файла на экране с помощью программы `xdvi` (`dvi`-драйверы не требуют специальной русификации и используются обычным образом) должны быть видны русские буквы, программа `dvips` должна создавать PostScript-файл, при просмотре и печати которого видны русские буквы и т. п.

Если всё это прошло успешно, то русификация готова к использованию (о некоторых ее возможностях см. приложение Е).

2. Windows-подобные системы

В этом разделе речь пойдет о том, как можно использовать LaTeX в рамках Windows. Прежде всего надо иметь в виду, что появляются всё новые и новые системы Microsoft Windows, так что приведённые здесь рекомендации могут не вполне подходить к тому, что установлено на вашем компьютере (мы надеемся, что они более или менее применимы к Windows 95/98/ME и Windows NT/2000/XP).

Существуют версии TeX для Windows, называемые fpTeX и MiKTeX. Обе они распространяются бесплатно, и выбор между ними — дело вкуса. Мы опишем установку и использование системы MiKTeX.

Заметим сразу же, что можно попробовать использовать и систему emTeX (о которой мы скажем несколько слов ниже) в режиме DOS-окна или DOS-эмуляции, хотя просмотр файлов на экране может быть менее удобным (а для просмотра PostScript-файлов всё равно понадобятся программы `Ghostscript` и `GSView`).

В отличие от UNIX, стандартный комплект Windows не включает в себя TeX'а и — надо полагать — не будет его включать в обозримом будущем (а в необозримом будущем не будет самой системы Windows). Поэтому нужно устанавливать не только русские добавления, но и весь TeX (несколько десятков мегабайтов), и надо иметь либо быстрый доступ к Интернет, либо компакт-диск. Для удобства читателей мы поместили комплект MiKTeX 2.0 (не самый новый, но вполне действующий) и русификацию, использованную при подготовке этой книги, в директорию `miktex.win` на прилагаемом к ней компакт-диске.

В этой директории есть файл `readme`, описывающий установку комплекта MiKTEX 2.0 с русскими добавлениями (а также Windows-версий программы `Ghostscript`). Прочтите его (скажем, в DOS-окне), даже если обычно вы не читаете инструкций — заодно убедитесь, что вы можете читать текстовые файлы в кодировке `cp866` (с которой придется иметь дело).

Установка состоит из четырех этапов.

1) Подготовка: распаковка файла с русскими добавлениями.

2) Установка MiKTEX 2.0 с помощью Setup Wizard (эта процедура аналогична установке других Windows-программ). При этом надо указать, что будут использоваться русские добавления (как именно это сделать, написано в файле `readme`). Мы предполагаем, что в системах типа Windows NT/2000/XP вам доступны права администратора.

3) Модификация переменной окружения `PATH`.

4) Установка русских добавлений (для этого нужно запустить установочный `bat`-файл из командной строки).

Программы `Ghostscript` и `GSView` устанавливаются обычным для Windows способом.

2.1. Установка системы MiKTEX с русскими добавлениями

Опишем процесс установки более подробно.

Помимо дистрибутива MiKTEX 2.0 (он находится в директории `miktex.win\miktex20`), используются файлы `zcmiktex.zip`, `unzip.exe` (программа разархивирования) и `test.tex` (тестовый файл для проверки работы установленной системы).

1) Разархивируйте файл `zcmiktex.zip` с сохранением структуры поддиректорий. Проще всего это сделать с помощью программы `unzip.exe`, списав ее вместе с `zcmiktex.zip` в какую-либо директорию и запустив в этой директории команду

   ```
   unzip.exe zcmiktex.zip
   ```

   Полученную директорию `zcmiktex` поместите в любое место жесткого диска (например, в `C:\`) и запомните ее адрес (например, `C:\zcmiktex`).

2) Начните установку MiKTEX с помощью SetupWizard, перейдя в директорию `miktex.win\miktex20\2.0` и запустив программу `setupwiz.exe`. Появится ряд экранов с вопросами:

   `Shared vs. Private Installation`
   [только для Windows NT/2000/XP и т. п.] Выберите «Install a shared MiKTEX environment».

   `Installation Directory`
   Укажите директорию `C:\zcmiktex\texmf` (с очевидными изменениями, если для `zcmiktex` было выбрано другое место). Предлагаемую по умолчанию директорию *не* используйте.

   `Component Selection`
   Если на диске достаточно места (необходимо примерно 100Mb), рекомендуется выбрать все компоненты. В любом случае необходимы LaTeX 2$_\varepsilon$ и LaTeX 2$_\varepsilon$ Packages, $\mathcal{AMS}$-LaTeX & $\mathcal{AMS}$Fonts, MetaPost и PSUtils.

   `Program Folder`
   Можно оставить то, что предлагается по умолчанию.

   `Local TEXMF Tree`
   Выберите «Create local TEXMF tree» и укажите

   ```
   C:\zcmiktex\localtexmf
   ```

   в качестве имени директории для генерируемых файлов (с очевидными изменениями, если для `zcmiktex` было выбрано другое место).

   `Additional TEXMF Directory Trees`
   Выберите «Incorporate preexisting TEXMF directory trees» и нажмите Next. На следующем экране нажмите Add и укажите директорию `C:\zcmiktex\zctexmf` (с очевидными изменениями, если для `zcmiktex` было выбрано другое место).

   После этого нажмите Next; начнется процесс установки. В конце будет предложено модифицировать переменную окружения `PATH`. От этого нужно отказаться.

   Закончите установку.

3) Добавьте

   ```
   C:\zcmiktex\texmf\miktex\bin
   ```

   и

```
C:\zcmiktex\zctexmf\miktex\bin
```

в переменную окружения `PATH` (с очевидными изменениями, если при установке вы указали другие имена). В Windows 95/98 это можно сделать, добавив в файл `autoexec.bat` строку типа

```
SET PATH=C:\zcmiktex\texmf\miktex\bin;C:\zcmiktex\zctexmf\miktex\bin;%PATH%
```

после других строк с `PATH`. В Windows NT/2000/XP нужно модифицировать переменную окружения `PATH` с помощью Control Panel → System → Advanced → Environment Variables. В Windows ME можно попробовать использовать программу `msconfig.exe`.

4) **Перезагрузите компьютер!**

5) Перейдите в директорию `C:\zcmiktex\zctexmf` (с очевидными изменениями, если было выбрано другое место для `zctexmf`) и введите в командной строке

```
install C:\zcmiktex\localtexmf C:\zcmiktex\zctexmf
```

(с очевидными изменениями, если нужно). После `localtexmf` и `zctexmf` *не должно быть* символа «\».

6) Установите обновления к MiKTEX 2.0. Для этого перейдите в директорию `miktex.win\miktex20\2.0-updates` и запустите программу `mt20up1.exe`. Проверьте, что в появившемся окне правильно указана директория, куда был установлен MiKTEX 2.0 (Installation Directory; если там что-то другое, это может быть признаком ошибки при выполнении вами предыдущих пунктов!), после чего нажмите кнопку Install. В конце работы программы нажмите кнопку Close.

2.2. Установка программ работы с PostScript-файлами

Установка GhostScript 7.05. Начните установку, перейдя в директорию `miktex.win\gs705` и запустив программу `gs705w32.exe`. В появившемся окне нажмите кнопку Setup.

В следующем окне «GNU Ghostscript Setup» укажите директорию, куда нужно установить программу `Ghostscript` (директория, имя которой содержит пробелы, типа «`Program Files`», не рекомендуется), и место, куда нужно поместить ярлыки (shortcuts) для программы `Ghostscript` и документации к ней. После этого нажмите кнопку Install.

Установка GSView. Лицензия, по которой распространяется программа GSView, не позволяет записать её на компакт-диск, прилагаемый к книге, хотя и разрешает записать её на компакт-диск, продаваемый (или раздаваемый) отдельно. Поэтому, если на имеющемся у вас диске программы `gsv43w32.exe` (в директории `miktex.win\gsview43`) не оказалось, спишите эту программу из Internet'а (она по нынешним меркам небольшая — меньше двух мегабайтов), например, с сервера `ftp.mccme.ru`, где в директории `pub/tex` лежит эта книга и материалы к ней.

Для установки `GSView` зайдите в директорию `miktex.win\gsview43` и запустите программу `gsv43w32.exe`. Диалог:

`Select Language`

Выберите язык для программы установки (например, English [английский])

`GSView Install`

Трижды нажмите кнопку Next. После этого выберите директорию для установки `GSView` (можно оставить то, что предлагают) и нажмите Next. Наконец, укажите место, куда нужно поместить ярлыки для программы `GSView` и документации к ней, и нажмите кнопку Finish.

Когда установка будет завершена, нажмите кнопку Exit для выхода из программы установки.

После установки можно опробовать полный цикл обработки файла в LaTeX'е.

2.3. Создание файла

Прежде всего надо создать файл (скажем, `test.tex`) в кодировке DOS (cp866) такого содержания:

```
\documentclass{article}
\begin{document}
This is a test file. Это~--- тестовый файл.
\end{document}
```

Такой файл надо создавать с помощью текстового редактора (отметим, что `WordPad`, `Notepad` и `Write` годятся не всегда). Например, можно применять редактор, встроенный в файловую оболочку типа `Far` (он обычно вызывается нажатием на клавишу F4); можно использовать программы типа Norton Commander или Volkov Commander в DOS-окне.

Если вы почему-либо используете текстовый редактор Microsoft Word, необходимо сохранить файл в формате типа «текст MS-DOS с концами строк» (в разных версиях Microsoft Word этот формат может называться по-разному). Проверьте, что полученный файл действительно текстовый и действительно в кодировке cp866. Это можно сделать, дав команду `type test.tex` в DOS-окне.

Существуют многие другие (freeware или shareware) программы-редакторы, которые на практике применяются для подготовки TEX-файлов (`WinEdt`, `Quickpad`, `WinEdit`, `UltraEdit`, `Aditor` и т. д.) — их можно найти в Интернете, но пользуясь ими, важно осознавать последствия своих действий (в какой кодировке записывается файл и др.).

2.4. Обработка файла с помощью LATEX'а

После того, как файл `test.tex` создан, надо дать команду

```
rlatex test.tex
```

На экране должно появиться примерно следующее:

```
This is TeX, Version 3.14159 (MiKTeX 2 UP 1)
(test.tex
This is Russified LaTeX2e [2000/07], based on LaTeX2e
2000/06/01 (C:\texmf\tex\latex\base\article.cls
Document Class: article 2000/05/19 v1.4b Standard LaTeX
document class (C:\texmf\tex\latex\base\size10.clo))
No file test.aux.
(C:\texmf\tex\latex\base\t1cmr.fd) [1] (test.aux) )
Output written on test.dvi (1 page, 284 bytes).
Transcript written on test.log.
```

(Такое сообщение появляется при первом запуске; при втором файл `test.aux` уже есть и на его отсутствие не жалуются.) При этом должны образоваться (помимо `test.aux`) еще файлы `test.dvi` и `test.log`. Последний файл должен содержать примерно то же самое, что было выведено на экран.

2.5. Просмотр и печать dvi-файла

Чтобы просмотреть файл `test.dvi` на экране, нужно дать команду

```
yap test.dvi
```

При этом должно появиться новое окно, в котором показана (возможно, не полностью) страница сверстанного текста. Клавиши Page Up и Page Down переходят на предыдущую и следующую страницу (пока, впрочем, страница только одна, и они не действуют), стрелки позволяют перемещаться по странице (что можно делать и с помощью мыши). Клавиши

«+» и «−» позволяют менять масштаб. Еще одна полезная возможность: набрав число и потом нажав клавишу `g`, можно перейти на страницу с заданным номером. Выйти из программы `yap` можно, нажав клавишу `q`.

С помощью меню File → Print можно также напечатать `dvi`-файл, если он не содержит вставок в формате PostScript. Если же такие вставки есть, нужно преобразовать `dvi`-файл в формат PostScript, после чего напечатать его с помощью программы GSView, как описано ниже.

Более подробно использование программы `yap` описано в документации.

2.6. Изготовление PostScript-файлов

Теперь следует опробовать работу с PostScript-файлами. Начнем с их создания: находясь всё в той же директории, где лежат файлы `test.tex` и `test.dvi`, дайте команду

```
dvips -o test.ps test.dvi
```

Программа `dvips` выдает на экран сообщение наподобие следующего:

```
This is dvips(k) 5.86 Copyright 1999 Radical Eye Software
(www.radicaleye.com) ' TeX output 2002.07.25:0115' -> test.ps
<texc.pro>. [1]
```

в котором [1] выдается при обработке первой страницы (для более длинных файлов страниц будет, естественно, больше). При этом создается файл `test.ps`.

Ради любопытства можно заглянуть в него с помощью текстового редактора: там написано что-то вроде

```
%!PS-Adobe-2.0
%%Creator: dvips(k) 5.86 Copyright 1999 Radical Eye Software
<...>
userdict /end-hook known{end-hook}if
%%EOF
```

(когда мы провели этот эксперимент, в файле оказалось 149 строк).

2.7. Просмотр и печать PostScript-файла

На практике, естественно, читать PostScript-файл невооруженным глазом не надо, а надо просматривать его с помощью программы `GSView`. Вызвать ее можно либо с помощью меню Start → Programs (тогда придется открывать файл с помощью клавиши «O» или пункта меню File →

Open), иногда достаточно щелкнуть мышью на файле `test.ps` в Windows Explorer или какой-нибудь файловой оболочке типа `Far`. (Если на машине установлены программы фирмы Adobe, они могут показывать на экране PostScript-файлы своим способом.)

При запуске программы `GSView` появится окно с сообщением, что программа не зарегистрирована; это не мешает работе программы, но может быть устранено с помощью регистрации (платной). После запуска программы в ее окне будет изображена первая (в данном случае и единственная) страница вашего файла. В этой программе также можно использовать клавиши Page Up и Page Down (или клавиши «+» и «−») для движения по страницам, клавиши «<» и «>» для изменения масштаба. Нажав клавишу P или выбрав в меню File → Print, можно напечатать файл (рекомендуем в диалоге печати установить «Windows GDI printer» — тогда можно будет устанавливать параметры печати обычным для Windows способом).

Отметим в заключение, что в комплект MiKTEX 2.0 входят разные полезные программы для обработки PostScript-файлов (программа `psselect` позволяет вырезать из файла отдельные страницы, программа `pstops` помещать несколько листов на один с уменьшением или без, программа `psbook` — печатать буклеты и т. п.; см. с. 407, только вместо команды `man` надо читать файлы с документацией).

Кроме того, в него входят различные программы обработки графических файлов в разных форматах, которые могут вам пригодиться при подготовке иллюстраций для включения в текст (коллекция программ `NetPBM`).

3. DOS-подобные системы

В системе DOS можно использовать пакет emTEX (автор — Eberhard Mattes). Он был написан довольно давно и включает в себя различные версии программ, рассчитанные на процессор 8086, 80286 и 80386 и выше. Его можно использовать с бо́льшим или меньшим успехом и в Windows в режиме DOS-эмуляции или в DOS-окне (возможно, потребуется что-то перенастраивать по части управления дополнительной памятью). Предупреждение: при одновременной установке emTEXи других TEX-систем на одном компьютере может произойти путаница (будут вызываться не те программы).

На прилагаемом компакт-диске имеются как исходные файлы (взятые с сервера CTAN, см. Ж.4), так и полный комплект установленных файлов вместе с добавлениями, сгенерированными форматами и др. Поэтому в оригинальные файлы можно не заглядывать, а воспользоваться этим комплектом. Как это сделать, описано в файле `emtex.dos/readme.em`.

Учтите, что этот комплект рассчитан на процессоры не ниже 386 и к тому же занимает довольно много места (несколько десятков мегабайтов), поскольку никакого отбора мы не делали и включили даже то, что вряд ли понадобится. На крайний случай — если ваш компьютер медленный и памяти мало — можно попробовать воспользоваться инсталляционной программой Виктора Хименко, которая написана в 1996 году и предназначена для установки emTEX'а и его русификации. Эта программа и соответствующие файлы из тогдашнего emTEX'а находятся в директории

`/justincase/emtexrus.old/1996`

Впрочем, это мы сообщаем лишь на крайний случай: во-первых, в этой программе есть ошибка, и она может отказываться работать на быстрых компьютерах или не находить нужных файлов. Во-вторых, изготовленные с помощью версии 1996 года `dvi`-файлы используют другие имена шрифтов и потому не будут обрабатываться теперешними `dvi`-драйверами.

4. Где взять дополнительные пакеты: архив CTAN

Мы старались включить в прилагаемый компакт-диск (содержимое и ISO-образ которого можно найти на сервере `ftp.mccme.ru`, директория `pub/tex`) всё необходимое для первоначальной установки LATEX'а (об одном исключении см. выше в разделе о `GSView`). Но через некоторое время вам скорее всего понадобится какой-то стилевой пакет или вспомогательная программа, которой на диске нет. (Или новая версия одного из пакетов.)

Стандартное место хранения TEX-материалов — архивы CTAN (Comprehensive TEX Archive Network). Главный сервер этой сети в настоящее время `ftp.dante.de`, см. также `www.dante.de`; в случае чего информацию об этой сети легко отыскать с помощью поисковых машин в Интернете.

Все материалы по TEX'у (за исключением материалов по использованной в книге русификации, которые можно найти на `ftp.mccme.ru`), помещенные на прилагаемый компакт-диск, были взяты именно из CTAN.

Большое количество русскоязычных материалов (версии русских шрифтов, разные вспомогательные пакеты) имеется на сервере Воронежского университета `ftp.vsu.ru`.

Приложение 3

Что читать дальше

Во-первых, полезно ознакомиться с книгой LaTeX Companion [6]: в ней приведено описание большого числа разнообразных LaTeX'овских стилевых пакетов плюс некоторые рекомендации по модификации оформления в духе нашей главы IX. Для профессиональных полиграфистов сведения, приведенные в книге [6] (и тем более в нашей книге), важны, но недостаточны: время от времени вы будете сталкиваться с загадочными сообщениями об ошибках или необычным поведением различных стилевых пакетов, не зная, как выправить ситуацию или внести в поведение пакета нужные вам модификации.

Если вы хотите изучить LaTeX глубже и обрести свободу в обращении с ним, то первое, что надо сделать, — прочитать TeXbook [2]: без знания того, как TeX работает «на низком уровне», дальше двигаться невозможно.

Одного этого, однако, мало: в книге [2] ни слова о LaTeX'е нет. В книге [1] вы найдете немало подробностей о том, как устроен LaTeX с точки зрения конечного пользователя (для нашей книги мы старались отобрать самое, на наш взгляд, полезное, но на полноту она не претендует), но про внутреннее устройство LaTeX'а там нет почти ничего. Узнать это внутреннее устройство можно единственным способом: изучая исходные тексты LaTeX'а и его стилевых пакетов. Все эти тексты доступны, и большая их часть снабжена комментариями (без этого разобраться в сколько-нибудь обширном наборе TeX'овских макросов весьма непросто).

Откомментированные исходные тексты LaTeX'овских пакетов хранятся в файлах с расширением `dtx` («`dtx`-файлах»), распространяющихся обычно вместе со стилевыми пакетами. Если обработать `dtx`-файл с помощью LaTeX'а, то получится `dvi`-файл, в котором обычно записано руководство по использованию пакета плюс исходный текст пакета с комментариями[1]. (Для fd-файлов, содержащих определения шрифтов, роль dtx-файлов играют так называемые fdd-файлы, обращаться с которыми надо так же, как с dtx-файлами.)

В некоторые установки TEX'а dtx-файлы не включаются с целью экономии места, но тогда обычно поставляются сгенерированные из них dvi-файлы. В любом случае все dtx- и fdd-файлы есть в Интернете на CTAN'е (см. приложение Ж).

Все компоненты ядра LATEX'а также доступны в dtx-формате и открыты для изучения. Остается только пожелать читателю успеха в этом непростом деле.

[1]Можно также обработать dtx-файл с помощью LATEX'а особым образом, и тогда из него получится sty-файл, который и представляет собой стилевой пакет.

Предметный указатель

E

F

Q

Литература

[1] L. Lamport. *LaTeX. A Document Preparation System, User's Guide and Reference Manual.* — Addison-Wesley, 1994. (Первое издание вышло в 1985 г; в нём описана система LaTeX 2.09.)

[2] D. E. Knuth. *The TeXbook,* часть A серии *Computers and Typesetting.* — Addison-Wesley, 1984. Русский перевод: Дональд Е. Кнут. *Все про TeX.* — Протвино: РДTeX, 1993.

[3] H. Partl, E. Schlegl, I. Hyna. *LaTeX-Kurzbeschreibung.* — Пособие в электронном виде; входит в состав пакета emTeX, доступного по адресу ftp.dante.de, directory tex-archive/systems/msdos/emtex

[4] Х. Партль, Э. Шлегль, И. Хина. *LaTeX: краткое описание.* — Пересказ с немецкого предыдущей версии пособия [3] с дополнениями А. Шеня (`shen@mccme.ru`). Рабочие материалы Независимого Московкого университета, 1993.

[5] M. Spivak. *The Joy of TeX. A gourmet guide to typesetting with the AMS-TeXmacro package.* — American Mathematical Society, Providence, RI, 1990. Русский перевод: М. Спивак. *Восхитительный TeX: руководство по комфортному изготовлению научных публикаций в пакете AMS-TeX.* — М.: Мир, 1993.

[6] M. Goossens, F. Mittelbach, A. Samarin. *The LaTeX Companion.* — Addison-Wesley, 1994. Русский перевод: М. Гуссенс, Ф. Миттельбах, А. Самарин. *Путеводитель по пакету LaTeX и его расширению LaTeX 2_ε.* Перевод с английского О. А. Маховой, Н. В. Третьякова, Ю. В. Тюменцева и В. В. Чистякова под редакцией И. А. Маховой. — М.: Мир, 1999.

[7] G. Grätzer. *Math into TeX. A simple introduction to AMS-LaTeX.* — Birkhäuser, 1993.

[8] G. Grätzer. *First steps in LaTeX.* Birkhäuser; Springer-Verlag, 1999. (ISBN 0-8176-4132-7, 3-7643-4132-7) Перевод: Г. Грэтцер. *Первые шаги в LaTeX'е.* Перевод с английского И. А. Маховой. — М.: Мир, 2000. — 172 с., илл. (ISBN 5-03-003366-1)

[9] И. А. Котельников, П. З. Чеботаев. *Издательская система LaTeX 2ε*. — Новосибирск: Сибирский хронограф, 1998.

[10] М. Гуссенс, С. Ратц. *Путеводитель по пакету LaTeX и его Web-приложениям.* При участии Э. М. Гурари, Р. Мура, С. Сьютора. Перевод с английского Ю. В. Тюменцева и А. В. Чернышёва под редакцией Б. В. Тоботраса. — М.: Мир, 2001. — 604 с., илл. (ISBN 5-030003387-4)

[11] М. Гуссенс, С. Ратц, Ф. Миттельбах. *Путеводитель по пакету LaTeX и его графическим расширениям. Иллюстрирование документов при помощи TeX'а и PostScript'а.* Перевод А. И. Лебедева и К. В. Мусатова под редакцией А. В. Лебедева. — М.: Мир: Бином ЛЗ, 2002. — 621 с., илл. (ISBN 5-03-003388-2 (Мир), 5-94774-027-3 (Бином ЛЗ))

[12] П. Каров. *Шрифтовые технологии. Описание и инструментарий.* Перевод с английского О. С. Карпинского и И. И. Куликова под редакцией, с предисловием и дополнением В. В. Ефимова. — М.: Мир, 2001. —454 с., илл.