



С. М. ЛЬВОВСКИЙ

Набор и верстка

в пакете

**L<sup>A</sup>T<sub>E</sub>X**

```
\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 \\
}
```

С. М. Львовский

# НАБОР И ВЁРСТКА в системе L<sup>A</sup>T<sub>E</sub>X

Издание пятое, переработанное

Москва  
Издательство МЦНМО  
2014

УДК 681.322  
ББК 32.97  
Л89

**Львовский С. М.**

Л89      Набор и вёрстка в системе  $\text{\LaTeX}$ . — 5-е изд., переработанное. — М.: МЦНМО, 2014. — 400 с.

ISBN 978-5-4439-0239-5

Книга посвящена популярной издательской системе  $\text{\LaTeX}$ , предназначенной для набора и верстки научно-технических текстов с формулами, таблицами, диаграммами любого уровня сложности. В настоящем издании книга существенно переработана: исключен ряд разделов, утративших актуальность, но добавлен материал о  $\text{\BibTeX}$ ’е,  $\text{\MetaPost}$ ’е, подготовке презентаций и пр.

Книга будет полезна всем, кто имеет дело с изготовлением на компьютере оригинал-макетов, а также авторам, самостоятельно набирающим научные тексты.

ББК 32.97

ISBN 978-5-4439-0239-5

© Львовский С. М., 2003, 2006, 2014.  
© МЦНМО, 2014.

# Оглавление

<b>Предисловие</b>	<b>6</b>
<b>I Элементарное введение</b>	<b>9</b>
1. Как проходит работа с системой $\text{\LaTeX}$ . . . . .	9
2. Основные понятия . . . . .	10
3. Набор формул в простейших случаях . . . . .	21
4. Обще $\text{\TeX}$ 'овские обозначения для выключных формул . .	26
5. Разбиение исходного файла на части . . . . .	27
6. Обработка ошибок . . . . .	29
7. Как читать книгу дальше? . . . . .	36
<b>II Как набирать формулы</b>	<b>38</b>
1. Таблицы спецзнаков с комментариями . . . . .	38
2. Важные мелочи . . . . .	51
3. Набор матриц . . . . .	63
4. Одно над другим . . . . .	67
5. Тонкая настройка . . . . .	80
<b>III Набор текста</b>	<b>88</b>
1. Специальные типографские знаки . . . . .	88
2. Подчеркивания, рамки . . . . .	91
3. Промежутки между словами . . . . .	91
4. Диакритические знаки и прочее . . . . .	94
5. Смена шрифтов в тексте . . . . .	96
6. Абзацы . . . . .	102
7. Специальные абзацы . . . . .	115
8. Сноски . . . . .	123
9. Между абзацами . . . . .	125
10. Линейки . . . . .	134
11. Для любознательных: абзацы нестандартной формы . . . .	137

<b>IV</b>	<b>Оформление текста в целом</b>	<b>141</b>
1.	Начнем с главного . . . . .	141
2.	Классы, пакеты и классовые опции . . . . .	142
3.	Стиль оформления страницы . . . . .	146
4.	Поля, размер страницы и прочее . . . . .	147
5.	Рубрикация документа . . . . .	151
6.	Эпиграфы . . . . .	156
7.	Титул, оглавление и пр. . . . .	158
8.	Предметный указатель . . . . .	162
9.	Иллюстрации и таблицы . . . . .	172
10.	Еще о метках и ссылках . . . . .	181
11.	Заметки на полях (маргиналии) . . . . .	184
<b>V</b>	<b>Печать текста с выравниванием</b>	<b>186</b>
1.	Окружение <code>tabbing</code> . . . . .	186
2.	Таблицы . . . . .	191
3.	Примеры . . . . .	198
4.	Дополнительные возможности . . . . .	205
<b>VI</b>	<b>Создание новых команд</b>	<b>213</b>
1.	Макроопределения . . . . .	213
2.	Счетчики . . . . .	226
3.	Окружения типа «теорема» . . . . .	235
4.	Параметры со значением длины . . . . .	239
5.	Создание новых окружений: общий случай . . . . .	242
<b>VII</b>	<b>Блоки и клей</b>	<b>246</b>
1.	Текст состоит из блоков . . . . .	246
2.	Команды $\text{\LaTeX}$ 'а для генерации блоков . . . . .	247
3.	Команда <code>\hbox</code> . . . . .	255
4.	Команда <code>\vbox</code> . . . . .	265
5.	Блочные переменные . . . . .	266
<b>VIII</b>	<b>Модификация стандартных классов</b>	<b>269</b>
1.	С чего начать . . . . .	270
2.	Снова о счетчиках . . . . .	272
3.	Рубрикация . . . . .	276
4.	Оглавление, список иллюстраций и прочее . . . . .	284
5.	Перечни общего вида . . . . .	292
6.	Колонтитулы . . . . .	299
7.	Плавающие объекты . . . . .	308

8. Разное . . . . .	314
---------------------	-----

## Приложения

А. Архитектура $\text{\TeX}$ 'а и $\text{\LaTeX}$ 'а . . . . .	320
Б. Библиографии в $\text{\LaTeX}$ 'е: $\text{\BibTeX}$ . . . . .	327
Г. Гиперссылки в pdf-файлах . . . . .	335
Д. Диаграммы (пакет $\text{\Xy-pic}$ ) . . . . .	337
И. Интернационализация $\text{\LaTeX}$ 'а . . . . .	343
К. Классы документов AMS . . . . .	353
М. Метапост . . . . .	355
О. Откуда взять $\text{\TeX}$ ? . . . . .	367
П. Презентации в $\text{\LaTeX}$ 'е . . . . .	369
Р. Рисунки с помощью подручных средств . . . . .	371
Ц. Цвет в $\text{\LaTeX}$ 'е . . . . .	375
Ч. Что читать дальше . . . . .	377

<b>Литература</b>	<b>379</b>
-------------------	------------

<b>Предметный указатель</b>	<b>381</b>
-----------------------------	------------

## Предисловие к пятому изданию

За 11 лет, прошедшие с выхода последнего переработанного издания этой книги, в компьютерном мире вообще и в  $\text{\TeX}$ 'овском мире в частности многое изменилось. Возможно, самое главное изменение состоит в том, что значительно возросли возможности поиска в интернете и объем содержащейся там информации. В связи с этим в новом издании я в целом ряде случаев опускаю подробности, ограничиваясь указанием на то, как их может найти заинтересованный читатель.

В отличие от предыдущих изданий, в этом я решил отдать предпочтение штатным средствам перед «самоделками»; в частности, полностью изгнаны рассказы о нестандартных русификациях  $\text{\LaTeX}$ 'а.

Книга подверглась не только сокращениям: в нее добавлен и новый материал, в частности рассказ о тесно связанных с  $\text{\LaTeX}$ 'ом системах Bib $\text{\TeX}$  и MetaPost.

В любом из изданий этой книги мой долг — выразить глубокую благодарность трем людям: А. Шеню, И. А. Маховой и В. Д. Арнольду. При подготовке настоящего издания мне были чрезвычайно полезны беседы с А. С. Мелик-Егановым, В. Ю. Радионовым (совместно с которым, в частности, написано приложение А) и В. В. Шуваловым, ценные советы М. Н. Вялого, Г. А. Мерзона и Д. Е. Щербакова (который также любезно согласился написать для настоящего издания приложение П) и нелицеприятная критика Т. Л. Коробковой. Этим своим друзьям и коллегам я с радостью говорю спасибо.

*С. Львовский*

## Предисловие

Замечательный программист и математик Дональд Кнут опередил свое время. Он разработал векторные шрифты (в формате METAFONT) за несколько лет до появления получивших всеобщее признание шрифтов в формате Postscript — но именно постскриптовские шрифты являются сейчас мейнстримом. Он разработал формат файлов (dvi — от DeVice Independent, независимый от устройства), предназначенный для представления сложных текстов в таком виде, чтоб их можно было просмотреть или напечатать под любой операционной системой, на любом компьютере или принтере, задолго до появления формата pdf — однако сейчас для этих целей используется именно pdf. А вот созданная им система компьютерной верстки  $\text{\TeX}$  (произносится «тех»), для нужд которой Кнут разрабатывал эти шрифты и этот формат файлов, и по сей

день жива, популярна и помирать не собирается, хотя по меркам компьютерного мира она очень стара: ее первая версия появилась еще в 1978 году, в 1989 году она приобрела современные очертания, и с тех пор мало изменилась.

Кое-что в  $\text{\TeX}$ 'е способно отпугнуть современного пользователя. В самом деле,  $\text{\TeX}$  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. Кроме того, чтобы в  $\text{\TeX}$ 'е работать, надо потратить определенное время на его изучение: трудно представить себе книгу под названием « $\text{\TeX}$  for dummies» (« $\text{\TeX}$  для болванов»).

А вот краткий перечень достоинств  $\text{\TeX}$ 'а.

- 1) Никакая другая из существующих в настоящее время издательских систем не может сравниться с  $\text{\TeX}$ 'ом в полиграфическом качестве текстов с математическими формулами.
- 2)  $\text{\TeX}$  реализован на всех современных компьютерных платформах, и все эти реализации действительно работают одинаково.
- 3) Благодаря этому  $\text{\TeX}$  стал международным языком для обмена математическими и физическими статьями: набрав свою статью в  $\text{\TeX}$ 'е, математик может послать ее по электронной почте своему коллеге или в редакцию журнала, даже если отправитель работает под Windows, а получатель — под UNIX'ом или, допустим, на Макинтоше.
- 4) Не знаю, уместно ли это называть достоинством, но при подаче статьи в большинство математических и физических журналов требуется набрать ее в  $\text{\TeX}$ 'е.
- 5) Наконец, основные реализации  $\text{\TeX}$ 'а для всех платформ распространяются бесплатно.

Разумеется, у  $\text{\TeX}$ 'а есть и недостатки. Главный из них — в том, что с помощью  $\text{\TeX}$ 'а тяжело готовить тексты со сложным расположением материала на странице (наподобие рекламных буклетов). Для таких приложений, практически не встречающихся в научно-технической литературе, лучше пользоваться другими программами.

Кроме того, — повторяю — в  $\text{\TeX}$ 'е 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.



Настоящее пособие посвящено популярной издательской системе, основанной на  $\text{T}_\text{E}\text{X}$ 'е, — системе  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ . В первую очередь оно предназначено для читателя, которому необходимо по роду своей работы готовить (писать или редактировать) тексты с формулами.

В идеале работа с  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 'ом должна бы выглядеть так: научный работник пишет на  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 'е (а редактор редактирует) текст с формулами, после чего файл поступает в опытные руки верстальщика, который доводит его до должного полиграфического качества. На практике автору или редактору порой приходится брать на себя часть «техредовской» работы. Имея это в виду, я рассказываю в книге несколько больше абсолютно необходимого минимума, так что книга может быть полезна и профессиональному полиграфисту, начинающему освоение  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 'а.

Эта книга выросла из попытки расширить и дополнить книгу [4] — выполненный А. Шенем пересказ с немецкого языка пособия [3] (с его дополнениями). Из исходного текста [3] позаимствованы макросы для печати  $\text{T}_\text{E}\text{X}$ 'овских примеров.

# Глава I

## Элементарное введение

### 1. Как проходит работа с системой ЛАТЭХ

Как уже отмечалось в предисловии, Т<sub>Ε</sub>X (произносится «тех», пишется также «TeX») — это созданная американским математиком и программистом Дональдом Кнудом (Donald E. Knuth) система для верстки текстов с формулами. Сам по себе Т<sub>Ε</sub>X 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 Т<sub>Ε</sub>X'a представляет собой пакет макроопределений (макропакет) этого языка. В частности, ЛАТЭХ (по-русски произносится «латех», пишется также «LaTeX») — это созданная Лесли Лэмпортом (Leslie Lamport) издательская система на базе Т<sub>Ε</sub>X'a.

Далее в этой книге мы будем, если не оговорено противного, употреблять слова Т<sub>Ε</sub>X и ЛАТЭХ вперемешку. При первом чтении можно воспринимать их как синонимы (на самом деле мы пишем Т<sub>Ε</sub>X, когда речь идет об общих свойствах систем на базе Т<sub>Ε</sub>X'a, а не о специфике именно ЛАТЭХ'a).

Для начала автор должен подготовить с помощью любого текстового редактора файл<sup>1</sup> с текстом, оснащенным командами для ЛАТЭХ'a. Такие файлы по традиции имеют расширение `tex` (описанию того, что должно быть в таком файле, и посвящена вся эта книга). Дальнейшая работа протекает в два этапа. Сначала надо обработать файл с помощью программы-транслятора. В современных версиях Т<sub>Ε</sub>X'a в результате трансляции получается `pdf`-файл. Кроме того, предусмотрена возможность получить файл с расширением `dvi` (device independent —

---

<sup>1</sup>Это должен быть именно «чистый» текстовый файл, а не `doc` или `rtf`.

не зависящий от устройства). В более старых версиях Т<sub>Е</sub>X’a генерация pdf-файлов не предусмотрена, но возможность создать dvi-файл была в Т<sub>Е</sub>X’e всегда.

С полученным pdf-файлом можно делать все то же, что обычно делают с pdf-файлами: просматривать, печатать, вести поиск и т. п. Если же получился dvi-файл, то его необходимо обрабатывать с помощью программ, называемых dvi-драйверами (и входящих в поставку Т<sub>Е</sub>X’a): распечатывать на принтере, смотреть на экране (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 Л<sub>А</sub>T<sub>Е</sub>X, вам необходимо уяснить для себя, что нужно сделать, чтобы оттранслировать исходный текст (т. е. создать из него pdf- или dvi-файл); если вы создаете именно dvi-файл, то надо также понять, что нужно сделать, чтобы просмотреть его на экране или напечатать. Кроме того, для создания исходного текста нужно, естественно, уметь обращаться с каким-нибудь текстовым редактором.

Во многих современных текстовых редакторах предусмотрена интеграция с Л<sub>А</sub>T<sub>Е</sub>X’ом, позволяющая проводить указанные выше манипуляции с tex- и dvi-файлами одним щелчком мыши или одной клавиатурной командой; в некоторых редакторах можно даже набирать наиболее частые Т<sub>Е</sub>X’овские команды с помощью меню, что, будь то к добру или к худу, приближает интерфейс такой интегрированной среды к интерфейсу редактора типа Word.

## 2. Основные понятия

### 2.1. Исходный файл

Исходный файл<sup>2</sup> для системы Л<sub>А</sub>T<sub>Е</sub>X представляет собой собственно текст документа вместе со *специмволами* 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 (Т<sub>Е</sub>X делает их сам). Слова отделяются друг от друга пробелами, при этом

---

<sup>2</sup>Для нетерпеливых: пример такого файла приведен на с. 14.

TeX не различает, сколько именно пробелов вы оставили между словами (чтобы вручную управлять пробелами, есть специальные команды, злоупотреблять которыми не советуем). Конец строки также воспринимается как пробел. Соседние абзацы должны быть отделены друг от друга пустыми строками (в произвольном количестве, 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 L<sup>A</sup>T<sub>E</sub>X.

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.

Как мог заметить читатель, тщательно форматировать исходный текст не обязательно.

## 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 фертинга повысился на 7%,  
и теперь за него дают \$200.

Курс фертинга 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.

Это % глупый  
% Лучше: поучительный  
пример.

Скажем вкратце о смысле остальных спецсимволов. Фигурные скобки ограничивают *группы* в исходном файле (см. с. 15). 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. 91). Со знака `\` начинаются все  $\TeX$ -овские команды (см. разд. 2.3). Знаки `#` и `&` используются в более сложных конструкциях, о которых сейчас говорить преждевременно.

Отметим еще, что символы `<` `>` `|` в тексте употреблять можно в том смысле, что сообщения об ошибке это не вызовет, но при этом может напечататься 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. Если вы пишете текст по-русски и оформляете его при этом должным образом (см. с. 29), то не употребляйте «просто так» и двойную кавычку `"` (она используется как вспомогательный символ для организации некоторых специфически русских полиграфических эффектов — см. разд. III.1.1 и III.1.2).

### 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 `\{`, `\}`, `\dots`, `\%`, о которых шла речь на с. 11.

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

Сразу же скажем, что использовать русские буквы в именах нельзя (если не предпринимать специальных ухищрений, говорить о которых мы не будем).

В именах команд прописные и строчные буквы различаются. Например, `\large`, `\Large` и `\LARGE` — это три разные команды (задающие, как мы потом увидим, 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:

`$1` или `$ 1?`

`\$1` или `\$ 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.`

`\slshape 2 turtle doves`

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

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

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 общий стиль оформления (как говорят, «класс документа»).

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. Кроме класса `book`, в стандартный комплект  $\text{\LaTeX}$ 'а входят классы `article`, `amsart` (для оформления статей), `report` (нечто среднее между `article` и `book`) и некоторые другие. 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 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 вам может захотеться их изменить). 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  $\text{\LaTeX}$  выдаст сообщение об ошибке. Часть файла, расположенная между командами `\documentclass` и `\begin{document}`, называется преамбулой.

Заканчиваться файл должен командой

```
\end{document}
```

Даже если после `\end{document}` в файле написано что-то еще,  $\text{\LaTeX}$  это проигнорирует.

Вот составленный по всем правилам  $\text{\LaTeX}$ -файл.

```
\documentclass{article}
\begin{document}
Hello world.
\end{document}
```

Если вы до сих пор не работали с системой  $\text{\LaTeX}$  на компьютере, сейчас стоит попробовать. Учтите только, что если вы собираетесь пробовать по-русски, то между `\documentclass` и `\begin{document}` надо обязательно написать кое-что еще — см. с. 29.

## 2.5. Группы

Важным понятием Т<sub>Е</sub>X'а является понятие *группы*. Чтобы понять, что это такое, рассмотрим пример.

При обработке Т<sub>Е</sub>X'ом исходного файла набор текста в каждый момент идет каким-то вполне определенным шрифтом (он называется текущим шрифтом). Команда `\slshape`, с которой мы уже столкнулись в разд. 2.3, переключает текущий шрифт на наклонный, а `\upshape` выполняет обратное переключение; команды `\bfseries` и `\mdseries` меняют жирность шрифта. Подробнее о переключении шрифтов будет рассказано в разделе III.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 Т<sub>Е</sub>X «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 на шрифт не влияют — они ограничивают *группу* внутри файла. Как правило, задаваемые командами Т<sub>Е</sub>X'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):



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; на печати это никак не отразилось, поскольку внутри скобок мы ничего не делали.

Трюк с постановкой пары скобок `{}` после имени команды, о котором шла речь на с. 13 — 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 (кроме скобок, входящих в состав команд `\{` и `\}`) должны быть сбалансированы: 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. Впрочем, глобальных команд в  $\text{\LaTeX}$ ’е мало, и дойдем мы до них нескоро.

## 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  $\text{\LaTeX}$ ’е аргументы команд бывают обязательные и необязательные. Обязательные аргументы задаются *в фигурных скобках*<sup>3</sup>; если для команды предусмотрено наличие обязательных аргументов, она без

<sup>3</sup>В некоторых случаях эти фигурные скобки можно опускать — см. с. 24.

них правильно работать не будет. У многих команд предусмотрены также и необязательные аргументы: они влияют на работу команды, коль 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<sup>4</sup>, и притом в две колонки, мы должны начать файл командой

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

Наряду с классовыми опциями в ЛАТ<sub>E</sub>X'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 стиливых 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  $sl_2(\mathbb{C})$ , а пакет `longtable` нужен, чтобы иметь возможность набирать таблицы, простирающиеся на несколько страниц. Когда мы говорим: «чтобы воспользоваться этой возможностью, необходимо подключить такие-то стиливые пакеты», мы молчаливо предполагаем, что поставка ЛАТ<sub>E</sub>X'а, которой вы пользуетесь, все эти пакеты содержит (важный пакет `amsfonts` в стандартную поставку входит).

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 ЛАТ<sub>E</sub>X'а — это окружение (environment).

*Окружение* — это фрагмент файла, начинающийся с текста

---

<sup>4</sup>Примечание для полиграфистов: Т<sub>E</sub>X'овский кегль 10 примерно соответствует нашему девятому кеглю (см. с. 19).

`\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:

<p>Все строки этого абзаца будут центрированы; переносов не будет, если только какое-то слово, как в дезоксирибонуклеиновой кислоте, не длинней строки.</p>	<pre>\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}</pre>
---	---

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 ЛАТЭХ'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, Т<sub>Е</sub>X в каждый момент обработки исходного текста учитывает значения различных

Таблица I.1. Т<sub>E</sub>X'овские единицы длины

pt	пункт $\approx 0.35$ мм
pc	пика = 12 pt
mm	миллиметр
cm	сантиметр = 10 мм
in	дюйм = 25,4 мм
dd	пункт Дидо $\approx 1,07$ pt
cc	цицero = 12 dd

параметров, таких, как величина абзацного отступа, ширина и высота страницы, расстояние по вертикали между соседними абзацами, а также великое множество других важных вещей. Расскажем, как можно менять эти параметры, если это понадобится.

Параметры Т<sub>E</sub>X'а обозначаются аналогично командам: с помощью символа \ («backslash»), за которым следует последовательность букв. Например, `\parindent` обозначает в Т<sub>E</sub>X'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 Т<sub>E</sub>X'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 Т<sub>E</sub>X'е при задании размеров.

Замечание для полиграфистов: Т<sub>E</sub>X'овский пункт является единицей измерения, принятой в англо-американской типографии; он равен  $1/72,27$  дюйма и тем самым отличается от пункта, принятого в континентальной Европе (в том числе и в России). Единица измерения, называемая в Т<sub>E</sub>X'е пунктом Дидо и равная  $1/72$  дюйма, 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  $\text{TeX}$ 'у, равна нулю, все равно необходимо указать при этом нуле какую-нибудь из используемых  $\text{TeX}$ 'ом единиц длины. Вместо `\parindent=0` надо писать `0pt` или `0in` (можно указать любую  $\text{TeX}$ 'овскую единицу длины).

Кроме перечисленных, в  $\text{TeX}$ 'е используются еще две «относительные» единицы длины, размер которых зависит от текущего шрифта. Это `em`, приблизительно равная ширине буквы «М» 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

$\text{LaTeX}$  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, но как же узнать заранее, каков будет ее номер, а узнав, как не запутаться, когда в процессе дальнейшей работы над текстом этот номер будет меняться? Чтобы избежать всех этих трудностей, надо 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.

Реализуется это 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 заведомо использовать любую последовательность букв (желательно латинских) 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:

Мойте руки.

Мойте руки.\label{wash}

Как известно (см. с. 21), руки надо мыть.

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  $\text{\LaTeX}$  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,  $\text{\LaTeX}$  сообщит вам об этом и попросит запустить программу еще раз, чтобы получить корректные ссылки. (На практике иногда нужно и больше запусков программы  $\text{\LaTeX}$  — 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» — пометить с возможностью ссылки можно почти любой элемент документа. Об этом мы будем подробно говорить в разд. IV.10.

### 3. Набор формул в простейших случаях

#### 3.1. Основные принципы

Формулами в  $\text{\TeX}$ е считаются как целые формулы, так и отдельные цифры или буквы, в том числе греческие. 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 в `LaTeX`'е заключаются между командами `\[` и `\]`, как в примере:

$$a^2 + b^2 = c^2$$

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, напротив, приходится указывать знаки препинания внутри формулы (до закрывающей ее команды `\]`), 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, в частности, увеличены расстояния между соседними буквами).

Использовать в формуле русские буквы в качестве переменных нельзя. Использовать русские буквы в текстовых включениях, задаваемых командами `\text` или `\mbox`, вполне можно (см. разд. II.2.4).

Часть файла, составляющая математическую формулу, образует группу (см. с. 15): 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

$$\backslash[ x^{\{4357\}}+y^{\{4357\}}=z^{\{4357\}} \backslash]$$

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  $R_{jkl}^i$  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  $R_j{}^i{}_{kl}$ .

Можно также написать  $R_{j\{\}\sim i\{\}\_kl}\$,.$

Если вы хотите написать формулу, читающуюся как «два в степени, равной  $x$  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  $x + 1/x \geq 2$  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  $\backslashge$ , «меньше или равно» — командой  $\backslashle$ ) (в гл. II рассказано, как получить знаки нестрогих неравенств в более привычном нам начертании  $\geq$  и  $\leq$ ). Символы  $<$  и  $>$  вне формул лучше не употреблять.

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 (иначе после нее будет оставлен небольшой дополнительный пробел, что нежелательно):

$$\pi \approx 3,14 \qquad \text{\texttt{\$}\pi\backslash approx 3\{,\}14\$}$$

Здесь команда `\pi` порождает греческую букву  $\pi$ , а команда `\approx` дает знак  $\approx$  («приблизительно 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:

$$\frac{(a+b)^2}{4} - \frac{(a-b)^2}{4} = ab \qquad \begin{array}{l} \text{\texttt{\$}[} \\ \text{\texttt{\quad \frac{(a+b)^2}{4}-} } \\ \text{\texttt{\quad \frac{(a-b)^2}{4}=} } \\ \text{\texttt{\quad ab} } \\ \text{\texttt{\$]} } \end{array}$$

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

$$\frac{1}{2} + \frac{x}{2} = \frac{1+x}{2} \qquad \begin{array}{l} \text{\texttt{\$[\frac{1}{2}+\frac{x}{2}=} } \\ \text{\texttt{\frac{1+x}{2}\$]} \end{array}$$

### 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:

$$1 + \left( \frac{1}{1-x^2} \right)^3 \qquad \begin{array}{l} \text{\texttt{\$[} } \\ \text{\texttt{\quad 1+\left(\frac{1}{1-x^2} } \\ \text{\texttt{\quad \right)^3} } \\ \text{\texttt{\$]} } \end{array}$$

Подробнее о разных размерах скобок 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):

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

Если надо записать формулу, читающуюся как « $x$  штрих в квадрате», возьмите  $x'$  в фигурные скобки, вот так:  $\{x'\}^2$  (иначе будет выдано сообщение об ошибке). Обратите еще внимание на то, что точку мы поставили в конце выключной формулы (если бы мы поставили ее после  $\backslash$ , 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  $1, 2, \dots, 100$ .

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  $\$1,2, \dots, 100\$$ .

Знак  $\sim$  после инициалов великого Гаусса мы поставили, чтобы фамилия не могла перенестись на другую строку отдельно от инициалов (см. с. 91). По той же причине поставлен знак  $\sim$  после начинающего предложение предлога «в». Команду `\ldots` можно использовать и в обычном тексте, вне математических формул, для знака многоточия (см. с. 90).

### 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  $\sin(\pi/6) = 1/2$ .  
`\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  $\operatorname{tg}$  и  $\operatorname{ctg}$  (в англоязычных странах эти функции принято обозначать  $\tan$  и  $\cot$  соответственно). Впрочем, если ваш русский  $\LaTeX$ 'овский файл оформлен корректно (см. с. 29, а также приложение И), то соответствующие команды будут определены. Подробности см. в разд. II.1.2.

## 4. Обще $\TeX$ 'овские обозначения для выключных формул

Наряду со специфически  $\LaTeX$ 'овским обозначением для выключных формул с помощью `\[` и `\]` можно использовать обще $\TeX$ 'овский способ: окружить выключную формулу парами знаков доллара `$$` с обеих сторон, как в примере:

$$a^2 + b^2 = c^2$$

`$$`  
`a^2+b^2=c^2`  
`$$`

В одном и том же файле можно использовать обозначения с помощью `$$ ... $$` и `\[ ... \]` вперемешку, но нельзя смешивать их в пределах одной выключной формулы: если вы начали выключную формулу с `$$`, то и «закрыть» ее надо парой долларов, а если вы открыли ее с помощью `\[`, то закрыть ее надо с помощью `\]`.

В большинстве случаев два способа обозначения выключных формул эквивалентны; однако же обозначение с помощью `\[` и `\]` в некоторых ситуациях упрощает смену оформления документа, а также дает на печати более удачные вертикальные интервалы вокруг выключной формулы. Рекомендуем в текстах, которые вы сами пишете «с нуля», пользоваться для выключных формул ЛАТЭХ'овскими обозначениями `\[` и `\]`.

## 5. 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 ЛАТЭХ'e предусмотрена команда `\input`. Если в тексте написать

```
\input{имя_файла}
```

то Т<sub>Э</sub>X 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 пяти глав, записанных в файлах `ch1.tex`, ..., `ch5.tex`, можно организовать в виде файла из восьми строчек (именно его, а не файлы с отдельными главами, надо будет передать для обработки ЛАТЭХ'у):

```
\documentclass[11pt]{report}
\begin{document}
\pagestyle{plain}
\input{ch1}
\input{ch2}
\input{ch3}
\input{ch4}
\input{ch5}
\end{document}
```

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

Ради реализма мы включили в преамбулу команд, означающую, что колонтитулов в вашей книге не будет, но будут колонцифры (номера страниц, печатающиеся внизу страницы). В гл. IV 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 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 использовании 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 и пятую главы вы еще не принялись, а вторую и третью уже всю редактируете. Тогда головной файл можно организовать так:

```
\documentclass[11pt]{report}
\pagestyle{plain}
\includeonly{ch2,ch3}
\begin{document}
\include{ch1}
\include{ch2}
\include{ch3}
\include{ch4}
\include{ch5}
\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.

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

## 6. Обработка ошибок

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

Все сообщения, которые  $\text{\TeX}$  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  $\text{\TeX}$ '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 (см. разд. III.6). Если, однако,  $\text{\TeX}$  натывается 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 16 строк<sup>5</sup>, в котором умышленно допущено несколько ошибок (только не сделайте лишних ошибок при наборе):

```
\documentclass{article}
\usepackage[cp1251]{inputenc}
\usepackage[russian]{babel}
\begin{document}
По-английски специалист по  $\text{\TeX}$ 'у называется  $\text{\TeX}$ pert.
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}=\lrf{1}{1+{\frac{1}{5}}\right)^2.
```

<sup>5</sup>Мы молчаливо предполагаем, что читатель набирает текст в кодировке `cp1251` («виндовой»). Если кодировка у вас другая, надо заменить `cp1251` на `cp866`, `koi8-r` или `utf8` в зависимости от используемой кодировки. Это один из способов начинать  $\text{\LaTeX}$ 'овский файл с русским текстом. Подробнее см. начало главы IV; по поводу других корректных способов оформления русского текста см. разд. II.5 приложения II.

\]

И последняя формула:  $\sqrt{4} = 2$ .

\end{document}

Теперь обработайте наш файл `test.tex` с помощью ЛАТ<sub>Э</sub>X'а. Вскоре вы увидите на экране вот что:

! Undefined control sequence.

1.5 ...алист по \TeX'у называется \TeXpert

?

Первая строка Т<sub>Э</sub>X'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). За строкой, начинающейся с восклицательного знака, всегда следует строка, начинающаяся с 1., после которой идет номер строки исходного текста с ошибкой (в нашем случае 5). После номера на экран выдается сама эта строка или та ее часть, которую Т<sub>Э</sub>X 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. 13), на которой Т<sub>Э</sub>X 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, и можно будет разобраться с ошибкой. Но можно и просто нажать клавишу «ввод»: при этом Т<sub>Э</sub>X 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 Т<sub>Э</sub>X'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 (Т<sub>Э</sub>X 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 7
                                ended by \end{centrr}.
See the LaTeX manual or LaTeX Companion for explanation.
Type H <return> for immediate help.
```

```
...
1.9 \end{centrr}
```

?

Это сообщение об ошибке начинается со слов **LaTeX Error**. Такого рода сообщения не встроены в  $\text{T}_{\text{E}}\text{X}$ , а создаются  $\text{\LaTeX}$ '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>
```

```
1.11 как (2x+1)~
```

$3=5x\$$ . И еще выключную формулу:

На сей раз мы забыли знак доллара, открывающий формулу;  $\text{T}_{\text{E}}\text{X}$ , однако, понял это не сразу, а лишь наткнувшись на символ  $\sim$ , который вне формул таким образом использовать нельзя. Нажмем «ввод»:  $\text{T}_{\text{E}}\text{X}$  исправит положение, мысленно вставив знак доллара непосредственно перед знаком  $\sim$ , и пойдет дальше (подчеркнем, что 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  $\text{T}_{\text{E}}\text{X}$  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.

1.12 `\[\frac{25}{36}=\lrfth`  
`(\frac{1}`

?

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

! Extra \right.

1.13 `{1+\frac{1}{5}}\right)`  
`^2`

?

Откуда это, ведь в строке 13 у нас все правильно?! Оказывается, эта ошибка была наведена предыдущей. В самом деле, перед этим Т<sub>Е</sub>X проигнорировал «команду» `\lrfth`, набранную вместо `\left` (именно так Т<sub>Е</sub>X и делает, если в ответ на ошибку «несуществующая команда» нажать на клавишу «ввод»), так что команду `\left` Т<sub>Е</sub>X вообще не видел; теперь выходит так, что в тексте, который видит Т<sub>Е</sub>X, 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; Т<sub>Е</sub>X опять по-свойски исправит ошибку, и вскоре вы увидите такое сообщение:

! Missing } inserted.

<inserted text>

}

<to be read again>

\$

1.15 И~последняя формула:  `$\sqrt{4} = 2$`

?

На сей раз ошибка в том, что мы забыли закрывающую фигурную скобку. Нажмем на «ввод»; Т<sub>Е</sub>X 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 Т<sub>Е</sub>X'a), после чего трансляция наконец завершится. Кстати, цифра 1 в квадратных скобках, появляющаяся при этом на экране, означает, что Т<sub>Е</sub>X сверстал страницу номер 1. 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 аргументе 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 — `null` 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»). Под UNIX-подобной операционной системой (например, `Linux`) в ответ на сообщение о такой ошибке можно и попросту нажать «ввод», и она будет проигнорирована.

Если команда `\input` с именем несуществующего файла попадетс $\TeX$ 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  $\TeX$  заинтересуется 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 вы 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 и случилось в нашем примере).

Наряду с пассивной реакцией на ошибки — все время нажимать на «ввод» или сказать **s** — можно 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 (`\lrf` вместо `\left`) наберем `i\left` и нажмем на «ввод». Следующей («наведенной») ошибки не будет: 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;  $\sqrt{4} = 2$  в нем останется, но не будет потеряно слово « $\TeX$ pert», 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  $\TeX$  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].

Сделать сто ошибок на короткой дистанции затруднительно: чаще всего сообщение о 100 ошибках свидетельствует о синтаксической ошибке, вызвавшей «зацикливание» программы. Нехватка памяти также обычно

возникает в результате заикливания, но иногда памяти может действительно не хватить. Так бывает, если в тексте встречаются чудовищно длинные абзацы<sup>6</sup> или сверхсложные таблицы с очень большим количеством строк и столбцов (см. гл. V по поводу таблиц). Если вы встретились с такой проблемой, то можно проконсультироваться со специалистом (или самому изучить по книге [2]), как использовать  $\TeX$  более эффективно, или попробовать найти транслятор  $\TeX$ 'а, дающий возможность работать с увеличенным объемом памяти.

В ответ на приглашение ? можно еще набрать  $\hbar$  или  $\mathbb{H}$  и нажать «ввод». В этом случае  $\TeX$  выдаст на экран дополнительную информацию по поводу вашей ошибки, а затем еще раз приглашение ?. Если вы не  $\TeX$ ник, то разобраться в этой информации будет непросто.

## 7. 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  $\LaTeX$  сам оформлял заголовки этих разделов и автоматически их нумеровал, как создать оглавление, и тому подобное.

В главе V рассказывается о печати таблиц с помощью  $\LaTeX$ 'а.

В главе VI объяснено, как можно повысить эффективность своей работы в  $\LaTeX$ 'е, создавая собственные команды. Первые разделы этой главы имеет смысл читать параллельно с главой II, да и вообще в эту главу полезно заглядывать параллельно с чтением более ранних глав.

Две последние главы предназначены прежде всего для читателей, интересующихся не только набором, но и версткой. В главе VII рассказывается о таких фундаментальных понятиях  $\TeX$ 'а, как «блоки» и «клей», а заключительная глава VIII, предполагающая знание всего

---

<sup>6</sup>В свое время на моем домашнем компьютере  $\TeX$ 'овская память иссякла, когда длина абзаца превысила 34 страницы. С современными компьютерами я таких экспериментов не проводил.

---

предыдущего материала, рассказывает, как изменить стандартный стиль оформления документов, предоставляемый L<sup>A</sup>T<sub>E</sub>X'ом, применительно к своим нуждам.

Книга завершается приложениями, посвященными более частным вопросам. Впрочем, некоторые из этих частных вопросов весьма важны.

## Глава II

# Как набирать формулы

О некоторых простейших приемах набора формул уже шла речь в первой главе, но для серьезной работы этого мало; 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 стилевых пакетов `amssymb` и `amsmath`; рекомендуем подключать их всегда, если в вашем тексте присутствуют сколько-нибудь сложные формулы. Напомним, что для того чтобы подключить, например, стилевые пакеты `amssymb` и `amsmath`, надо в преамбулу документа следует включить строчку вида

```
\usepackage{amssymb,amsmath}
```

Имея все это в виду, приступим к делу.

### 1. Таблицы спецзнаков с комментариями

В этом разделе мы перечислим математические знаки, предоставляемые L<sup>A</sup>T<sub>E</sub>X'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  $\alpha$  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.

$\alpha$	<code>\alpha</code>	$\beta$	<code>\beta</code>	$\gamma$	<code>\gamma</code>
$\delta$	<code>\delta</code>	$\epsilon$	<code>\epsilon</code>	$\varepsilon$	<code>\varepsilon</code>
$\zeta$	<code>\zeta</code>	$\eta$	<code>\eta</code>	$\theta$	<code>\theta</code>
$\vartheta$	<code>\vartheta</code>	$\iota$	<code>\iota</code>	$\kappa$	<code>\kappa</code>
$\lambda$	<code>\lambda</code>	$\mu$	<code>\mu</code>	$\nu$	<code>\nu</code>
$\xi$	<code>\xi</code>	$\pi$	<code>\pi</code>	$\varpi$	<code>\varpi</code>
$\rho$	<code>\rho</code>	$\varrho$	<code>\varrho</code>	$\sigma$	<code>\sigma</code>
$\varsigma$	<code>\varsigma</code>	$\tau$	<code>\tau</code>	$\upsilon$	<code>\upsilon</code>
$\phi$	<code>\phi</code>	$\varphi$	<code>\varphi</code>	$\chi$	<code>\chi</code>
$\psi$	<code>\psi</code>	$\omega$	<code>\omega</code>		

Имя команды, задающей прописную греческую букву, пишется с прописной буквы (например, буква  $\Psi$  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. 54 по поводу того, как это сделать). Не надо использовать греческие буквы  $\Sigma$  и  $\Pi$  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:

$\Gamma$	<code>\Gamma</code>	$\Delta$	<code>\Delta</code>	$\Theta$	<code>\Theta</code>
$\Lambda$	<code>\Lambda</code>	$\Xi$	<code>\Xi</code>	$\Pi$	<code>\Pi</code>
$\Sigma$	<code>\Sigma</code>	$\Upsilon$	<code>\Upsilon</code>	$\Phi$	<code>\Phi</code>
$\Psi$	<code>\Psi</code>	$\Omega$	<code>\Omega</code>		

Читатель мог заметить, что прописные греческие буквы печатаются, в отличие от строчных, прямым шрифтом. Если вам нужны наклонные прописные греческие буквы (вроде  $\Sigma$ ), то проще всего подключить пакет `amssymb` (тем более что он вам почти наверняка понадобится и для других целей) — тогда наклонные греческие буквы можно будет набирать следующим образом:

$\Gamma$	<code>\varGamma</code>	$\Delta$	<code>\varDelta</code>	$\Theta$	<code>\varTheta</code>
$\Lambda$	<code>\varLambda</code>	$\Xi$	<code>\varXi</code>	$\Pi$	<code>\varPi</code>
$\Sigma$	<code>\varSigma</code>	$\Upsilon$	<code>\varUpsilon</code>	$\Phi$	<code>\varPhi</code>
$\Psi$	<code>\varPsi</code>	$\Omega$	<code>\varOmega</code>		



Если пакет `amssymb` почему-то не подключен, то есть альтернативный способ получить наклонную прописную греческую букву, о котором будет сказано в разд. 2.3.

Следующая серия символов — символы, рассматриваемые Т<sub>Е</sub>X'ом как символы бинарных операций (наподобие знаков сложения, умножения и т. п.); Т<sub>Е</sub>X 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:

$+$	<code>+</code>	$-$	<code>-</code>	$*$	<code>*</code>
$\pm$	<code>\pm</code>	$\mp$	<code>\mp</code>	$\times$	<code>\times</code>
$\div$	<code>\div</code>	$\setminus$	<code>\setminus</code>	$\cdot$	<code>\cdot</code>
$\circ$	<code>\circ</code>	$\bullet$	<code>\bullet</code>	$\cap$	<code>\cap</code>
$\cup$	<code>\cup</code>	$\uplus$	<code>\uplus</code>	$\sqcap$	<code>\sqcap</code>
$\sqcup$	<code>\sqcup</code>	$\vee$	<code>\vee</code>	$\wedge$	<code>\wedge</code>
$\oplus$	<code>\oplus</code>	$\ominus$	<code>\ominus</code>	$\otimes$	<code>\otimes</code>
$\odot$	<code>\odot</code>	$\oslash$	<code>\oslash</code>	$\triangleleft$	<code>\triangleleft</code>
$\triangleright$	<code>\triangleright</code>	$\amalg$	<code>\amalg</code>	$\diamond$	<code>\diamond</code>
$\wr$	<code>\wr</code>	$\star$	<code>\star</code>	$\dagger$	<code>\dagger</code>
$\ddagger$	<code>\ddagger</code>	$\triangleup$	<code>\triangleup</code>	$\bigcirc$	<code>\bigcirc</code>
$\nabla$	<code>\nabla</code>				

Обозначения для многих из выписанных знаков длинны и сложны. Если в вашем тексте часто встречается математический символ с длинным обозначением, имеет смысл определить для него свою более удобную команду. Если, например, хочется иметь возможность писать `\btu` вместо `\bigtriangleup`, включите в свой файл (после всех команд `\usepackage`, но до `\begin{document}`) такую строку:

```
\newcommand*\btu{\bigtriangleup}
```

Подробности см. в гл. VI; вы можете почитать ее уже сейчас.

При подключении пакета `amssymb` можно также воспользоваться следующими дополнительными математическими знаками для бинарных операций, разработанными Американским математическим обществом (сокращенно AMS — American Mathematical Society).

$\boxdot$	<code>\boxdot</code>	$\boxplus$	<code>\boxplus</code>	$\boxtimes$	<code>\boxtimes</code>
$\centerdot$	<code>\centerdot</code>	$\boxminus$	<code>\boxminus</code>	$\veebar$	<code>\veebar</code>
$\barwedge$	<code>\barwedge</code>	$\doublebarwedge$	<code>\doublebarwedge</code>	$\Cup$	<code>\Cup</code>
$\Cap$	<code>\Cap</code>	$\curlywedge$	<code>\curlywedge</code>	$\curlyvee$	<code>\curlyvee</code>
$\leftthreetimes$	<code>\leftthreetimes</code>	$\rightthreetimes$	<code>\rightthreetimes</code>	$\dotplus$	<code>\dotplus</code>

$\intercal$	<code>\intercal</code>	$\odot$	<code>\circledcirc</code>	$\circledast$	<code>\circledast</code>
$\ominus$	<code>\circleddash</code>	$\div$	<code>\divideontimes</code>	$\lessdot$	<code>\lessdot</code>
$\gtrdot$	<code>\gtrdot</code>	$\ltimes$	<code>\ltimes</code>	$\rtimes$	<code>\rtimes</code>
$\smallsetminus$	<code>\smallsetminus</code>				

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.

$<$	<code>&lt;</code>	$>$	<code>&gt;</code>	$=$	<code>=</code>
$:$	<code>:</code>	$\leq$	<code>\le</code>	$\geq$	<code>\ge</code>
$\neq$	<code>\ne</code>	$\sim$	<code>\sim</code>	$\simeq$	<code>\simeq</code>
$\approx$	<code>\approx</code>	$\cong$	<code>\cong</code>	$\equiv$	<code>\equiv</code>
$\ll$	<code>\ll</code>	$\gg$	<code>\gg</code>	$\doteq$	<code>\doteq</code>
$\parallel$	<code>\parallel</code>	$\perp$	<code>\perp</code>	$\in$	<code>\in</code>
$\notin$	<code>\notin</code>	$\ni$	<code>\ni</code>	$\subset$	<code>\subset</code>
$\subseteq$	<code>\subseteq</code>	$\supset$	<code>\supset</code>	$\supseteq$	<code>\supseteq</code>
$\succ$	<code>\succ</code>	$\prec$	<code>\prec</code>	$\succeq$	<code>\succeq</code>
$\preceq$	<code>\preceq</code>	$\asymp$	<code>\asymp</code>	$\sqsubseteq$	<code>\sqsubseteq</code>
$\sqsupseteq$	<code>\sqsupseteq</code>	$\models$	<code>\models</code>	$\vdash$	<code>\vdash</code>
$\dashv$	<code>\dashv</code>	$\smile$	<code>\smile</code>	$\frown$	<code>\frown</code>
$\mid$	<code>\mid</code>	$\bowtie$	<code>\bowtie</code>	$\propto$	<code>\propto</code>

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

Команда `\mid` в нашей таблице определяет вертикальную черточку, рассматриваемую как знак бинарного отношения. Типичный случай, когда она нужна — запись определения множеств:

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

Если тут написать `|` вместо `\mid`, то пробелы вокруг вертикальной черты будут недостаточны. С другой стороны, и команду `\mid` не следует употреблять, если вертикальная черточка употребляется как аналог

скобки (например, как знак абсолютной величины). Команды `\`, нужны, чтобы сделать дополнительные маленькие пробелы возле фигурных скобок (подробнее см. разд. II.5). (Если вам жаль времени на эти дополнительные пробелы, не делайте их: для черновых версий сойдет и так, а в беловике это сделает технический редактор.)

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

$$f: X \rightarrow Y \qquad \text{\texttt{\$f\colon X\to Y\$}}$$

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

При подключении пакета `amssymb` доступных символов для бинарных отношений станет гораздо больше. Вот что предлагает этот пакет сверх стандартного L<sup>A</sup>T<sub>E</sub>X'овского набора.

$\Rrightarrow$	<code>\rightleftharpoons</code>	$\Lrightarrow$	<code>\leftrightharpoons</code>
$\Vdash$	<code>\Vdash</code>	$\Vdash$	<code>\Vdash</code>
$\vDash$	<code>\vDash</code>	$\Uparrow$	<code>\upharpoonright</code>
$\downharpoonright$	<code>\downharpoonright</code>	$\upharpoonleft$	<code>\upharpoonleft</code>
$\downharpoonleft$	<code>\downharpoonleft</code>	$\Lsh$	<code>\Lsh</code>
$\Rsh$	<code>\Rsh</code>	$\circ$	<code>\circeq</code>
$\succsim$	<code>\succsim</code>	$\gtrsim$	<code>\gtrsim</code>
$\gtrapprox$	<code>\gtrapprox</code>	$\multimap$	<code>\multimap</code>
$\therefore$	<code>\therefore</code>	$\because$	<code>\because</code>
$\doteqdot$	<code>\doteqdot</code>	$\triangleq$	<code>\triangleq</code>
$\precsim$	<code>\precsim</code>	$\lesssim$	<code>\lesssim</code>
$\lessapprox$	<code>\lessapprox</code>	$\eqslantless$	<code>\eqslantless</code>
$\eqslantgtr$	<code>\eqslantgtr</code>	$\curlyeqprec$	<code>\curlyeqprec</code>
$\curlyeqsucc$	<code>\curlyeqsucc</code>	$\preccurlyeq$	<code>\preccurlyeq</code>
$\leqq$	<code>\leqq</code>	$\leqslant$	<code>\leqslant</code>
$\lessgtr$	<code>\lessgtr</code>	$\risingdotseq$	<code>\risingdotseq</code>
$\fallingdotseq$	<code>\fallingdotseq</code>	$\succcurlyeq$	<code>\succcurlyeq</code>
$\geqq$	<code>\geqq</code>	$\geqslant$	<code>\geqslant</code>
$\gtrless$	<code>\gtrless</code>	$\sqsubset$	<code>\sqsubset</code>
$\sqsupset$	<code>\sqsupset</code>	$\vartriangleright$	<code>\vartriangleright</code>
$\vartriangleleft$	<code>\vartriangleleft</code>	$\trianglerighteq$	<code>\trianglerighteq</code>
$\trianglelefteq$	<code>\trianglelefteq</code>	$\between$	<code>\between</code>
$\blacktriangleright$	<code>\blacktriangleright</code>	$\blacktriangleleft$	<code>\blacktriangleleft</code>
$\vartriangle$	<code>\vartriangle</code>	$\eqcirc$	<code>\eqcirc</code>
$\lesseqgtr$	<code>\lesseqgtr</code>	$\gtreqless$	<code>\gtreqless</code>
$\lesseqqgtr$	<code>\lesseqqgtr</code>	$\gtreqqless$	<code>\gtreqqless</code>

$\propto$	<code>\varpropto</code>	$\smile$	<code>\smallsmile</code>
$\frown$	<code>\smallfrown</code>	$\subseteq$	<code>\Subset</code>
$\supset$	<code>\Supset</code>	$\supseteq$	<code>\supseteqq</code>
$\supseteq$	<code>\supseteqeq</code>	$\bumpeq$	<code>\bumpeq</code>
$\Bumpeq$	<code>\Bumpeq</code>	$\lll$	<code>\lll</code>
$\ggg$	<code>\ggg</code>	$\pitchfork$	<code>\pitchfork</code>
$\backsim$	<code>\backsim</code>	$\backsimeq$	<code>\backsimeq</code>
$\lvertneqq$	<code>\lvertneqq</code>	$\gvertneqq$	<code>\gvertneqq</code>
$\lneqq$	<code>\lneqq</code>	$\gneqq$	<code>\gneqq</code>
$\lneq$	<code>\lneq</code>	$\gneq$	<code>\gneq</code>
$\precnsim$	<code>\precnsim</code>	$\succnsim$	<code>\succnsim</code>
$\lnsim$	<code>\lnsim</code>	$\gnsim$	<code>\gnsim</code>
$\precneqq$	<code>\precneqq</code>	$\succneqq$	<code>\succneqq</code>
$\precnapprox$	<code>\precnapprox</code>	$\succnapprox$	<code>\succnapprox</code>
$\lnapprox$	<code>\lnapprox</code>	$\gnapprox$	<code>\gnapprox</code>
$\varsubsetneq$	<code>\varsubsetneq</code>	$\varsupsetneq$	<code>\varsupsetneq</code>
$\subsetneqq$	<code>\subsetneqq</code>	$\supsetneqq$	<code>\supsetneqq</code>
$\varsubsetneqq$	<code>\varsubsetneqq</code>	$\varsupsetneqq$	<code>\varsupsetneqq</code>
$\subsetneq$	<code>\subsetneq</code>	$\supsetneq$	<code>\supsetneq</code>
$\eqsim$	<code>\eqsim</code>	$\shortmid$	<code>\shortmid</code>
$\shortparallel$	<code>\shortparallel</code>	$\thicksim$	<code>\thicksim</code>
$\thickapprox$	<code>\thickapprox</code>	$\approxeq$	<code>\approxeq</code>
$\succapprox$	<code>\succapprox</code>	$\precapprox$	<code>\precapprox</code>
$\backepsilon$	<code>\backepsilon</code>		

Знаки  $\leq$  и  $\geq$  из этой таблицы нам уже знакомы.

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:

$\nleq$	<code>\nleq</code>	$\ngeq$	<code>\ngeq</code>	$\nless$	<code>\nless</code>
$\ngtr$	<code>\ngtr</code>	$\nprec$	<code>\nprec</code>	$\nsucc$	<code>\nsucc</code>
$\nleqslant$	<code>\nleqslant</code>	$\ngeqslant$	<code>\ngeqslant</code>	$\npreceq$	<code>\npreceq</code>
$\nsucceq$	<code>\nsucceq</code>	$\nleqq$	<code>\nleqq</code>	$\ngeqq$	<code>\ngeqq</code>
$\nsim$	<code>\nsim</code>	$\ncong$	<code>\ncong</code>	$\nsubseteqq$	<code>\nsubseteqq</code>
$\nsupseteqq$	<code>\nsupseteqq</code>	$\nsubseteqq$	<code>\nsubseteqq</code>	$\nsupseteq$	<code>\nsupseteq</code>
$\nparallel$	<code>\nparallel</code>	$\nmid$	<code>\nmid</code>	$\nshortmid$	<code>\nshortmid</code>
$\nshortparallel$	<code>\nshortparallel</code>	$\nvdash$	<code>\nvdash</code>	$\nVdash$	<code>\nVdash</code>
$\nvDash$	<code>\nvDash</code>	$\nVDash$	<code>\nVDash</code>	$\ntrianglerighteq$	<code>\ntrianglerighteq</code>

$\ntrianglelefteq$	<code>\ntrianglelefteq</code>	$\ntriangleleft$	<code>\ntriangleleft</code>	$\ntriangleright$	<code>\ntriangleright</code>
$\nleftarrow$	<code>\nleftarrow</code>	$\nrightarrow$	<code>\nrightarrow</code>	$\nLeftarrow$	<code>\nLeftarrow</code>
$\nrightarrow$	<code>\nrightarrow</code>	$\nLeftrightarrow$	<code>\nLeftrightarrow</code>	$\nleftrightarrow$	<code>\nleftrightarrow</code>

В следующей таблице собраны стрелки различных видов (с точки зрения Т<sub>Е</sub>X'a, стрелки — это тоже знаки бинарных отношений, но математики, как правило, так не считают).

$\rightarrow$	<code>\to</code>	$\longrightarrow$	<code>\longrightarrow</code>	$\Rightarrow$	<code>\Rightarrow</code>
$\Longrightarrow$	<code>\Longrightarrow</code>	$\hookrightarrow$	<code>\hookrightarrow</code>		
$\mapsto$	<code>\mapsto</code>	$\longmapsto$	<code>\longmapsto</code>		
$\leftarrow$	<code>\gets</code>	$\longleftarrow$	<code>\longleftarrow</code>	$\Leftarrow$	<code>\Leftarrow</code>
$\Longleftarrow$	<code>\Longleftarrow</code>	$\hookleftarrow$	<code>\hookleftarrow</code>		
$\leftrightarrow$	<code>\leftrightarrow</code>	$\longleftrightarrow$	<code>\longleftrightarrow</code>		
$\Leftrightarrow$	<code>\Leftrightarrow</code>	$\Longleftrightarrow$	<code>\Longleftrightarrow</code>		
$\uparrow$	<code>\uparrow</code>	$\Uparrow$	<code>\Uparrow</code>		
$\downarrow$	<code>\downarrow</code>	$\Downarrow$	<code>\Downarrow</code>		
$\updownarrow$	<code>\updownarrow</code>	$\Updownarrow$	<code>\Updownarrow</code>		
$\nearrow$	<code>\nearrow</code>	$\searrow$	<code>\searrow</code>		
$\swarrow$	<code>\swarrow</code>	$\nwarrow$	<code>\nwarrow</code>	$\leftharpoonup$	<code>\leftharpoonup</code>
$\leftharpoonup$	<code>\leftharpoonup</code>	$\rightharpoonup$	<code>\rightharpoonup</code>		
$\rightharpoonup$	<code>\rightharpoonup</code>	$\rightleftharpoons$	<code>\rightleftharpoons</code>		

Если подключить пакет `amssymb`, то, как водится, доступных стрелок станет больше:

$\circlearrowright$	<code>\circlearrowright</code>	$\circlearrowleft$	<code>\circlearrowleft</code>
$\twoheadrightarrow$	<code>\twoheadrightarrow</code>	$\twoheadleftarrow$	<code>\twoheadleftarrow</code>
$\leftrightsquigarrow$	<code>\leftrightsquigarrow</code>	$\rightleftarrows$	<code>\rightleftarrows</code>
$\upuparrows$	<code>\upuparrows</code>	$\downdownarrows$	<code>\downdownarrows</code>
$\rightarrowtail$	<code>\rightarrowtail</code>	$\leftarrowtail$	<code>\leftarrowtail</code>
$\leftrightarrows$	<code>\leftrightarrows</code>	$\rightleftarrows$	<code>\rightleftarrows</code>
$\rightsquigarrow$	<code>\rightsquigarrow</code>	$\leftrightsquigarrow$	<code>\leftrightsquigarrow</code>
$\looparrowleft$	<code>\looparrowleft</code>	$\looparrowright$	<code>\looparrowright</code>
$\Rrightarrow$	<code>\Rrightarrow</code>	$\Lleftarrow$	<code>\Lleftarrow</code>
$\nleftarrow$	<code>\nleftarrow</code>	$\nrightarrow$	<code>\nrightarrow</code>
$\nLeftarrow$	<code>\nLeftarrow</code>	$\nrightarrow$	<code>\nrightarrow</code>
$\nLeftrightarrow$	<code>\nLeftrightarrow</code>	$\nleftrightarrow$	<code>\nleftrightarrow</code>
$\curvearrowleft$	<code>\curvearrowleft</code>	$\curvearrowright$	<code>\curvearrowright</code>

Среди AMS-овских команд для стрелок есть и `\rightleftharpoons`, входящая в основной набор Л<sup>A</sup>T<sub>Е</sub>X'a; ниже можно найти еще несколько аналогичных примеров. Такое дублирование — не прихоть: в базовом Л<sup>A</sup>T<sub>Е</sub>X'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  $\Rightarrow$  (`\rightleftharpoons`) или, скажем,  $\hbar$  (`\hbar`) правильно ведут себя и в индексах.

## 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. 26).

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

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 русских текстах эти функции, как известно, принято обозначать по-другому. Однако же если вы пишете в ЛАТ<sub>E</sub>X’е по-русски, то, как мы уже отмечали на с. 29 и еще не раз отметим, в начале файла сразу после `\documentclass` необходимо написать

```
\usepackage[кодировка]{inputenc}
\usepackage[russian]{babel}
```

где *кодировка* — это `cp1251`, `koi8-r`, `cp866` или `utf8` (в зависимости от того, какой русской кодировкой вы пользуетесь)<sup>1</sup>. После того как это сделано, в вашем распоряжении окажутся следующие девять команд, задающие привычные отечественному читателю обозначения для тригонометрических и им подобных функций:

<code>\tg</code>	<code>\ctg</code>	<code>\arctg</code>
<code>\arcctg</code>	<code>\sh</code>	<code>\ch</code>
<code>\th</code>	<code>\cth</code>	<code>\cosec</code>

<sup>1</sup>Если ответа на этот вопрос вы не знаете, а работаете при этом в системе Windows, то скорее всего ваш редактор сохраняет файлы в кодировке `cp1251`. В разд. II.5 приложения И описаны другие корректные способы оформления русского текста, но в любом из них подключается пакет `babel`, содержащий «русские» обозначения для тригонометрических функций.

Обозначения наподобие вышеперечисленных встречаются в математике очень часто, заготовить их на все случаи жизни все равно невозможно, поэтому нередко приходится такие обозначения определять самому. Это просто: надо подключить пакет `amsmath`, после чего добавить в преамбулу такую строчку:

```
\DeclareMathOperator{\Ext}{Ext}
```

В результате, например, запись `\Ext^2(A,B)` будет давать на печати  $\text{Ext}^2(A, B)$ . В первом аргументе команды `\DeclareMathOperator` ставится придуманное вами имя команды (если окажется, что оно уже занято,  $\text{\LaTeX}$  зафиксирует ошибку), 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  $\text{\LaTeX}$ е такие команды есть, но на печати они дают не `Re` и `Im`, а  $\Re$  и  $\Im$  (см. с. 49), 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  $\text{\LaTeX}$ е конструкции для (пере)определения новых команд (см. гл. VI).

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:

$$23^{1993} \equiv 1 \pmod{11}. \qquad \$23^{\sim{1993}}\equiv 1\pmod{11}$.$$

Обратите внимание, что скобки вокруг  $\pmod{11}$  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 `\pmod`, другими способами:

$$\begin{array}{ll} a^{p-1} \equiv 1 \pmod{p} & \$a^{\sim{p-1}}\equiv 1\mod p\$ \\ a^{p-1} \equiv 1 \pmod{p} & \$a^{\sim{p-1}}\equiv 1\pod p\$ \end{array}$$

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

$$f_*(x) = f(x) \mod G \qquad \$f_*(x)=f(x)\bmod G\$$$

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. Эти элементы, называемые «пределами суммирования» (по-английски *limits*), в исходном тексте обозначаются точно так же, как индексы; имея в виду, что знак суммы генерируется командой `\sum`, заключаем, что вышеназванную формулу можно получить так:

$$\begin{array}{l} \backslash[ \\ \quad \backslash\sum_{i=1}^{\sim n} n^2=\backslashfrac{n(n+1)(2n+1)}{6} \\ \backslash] \end{array}$$

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

$$\sum_{i=1}^n (2n-1) = n^2 \qquad \$\backslash\sum_{i=1}^{\sim n} (2n-1)=n^2\$$$

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



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

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

$\overline{\lim}$	<code>\varlimsup</code>	$\underline{\lim}$	<code>\varliminf</code>
$\operatorname{inj\,lim}$	<code>\injlim</code>	$\operatorname{proj\,lim}$	<code>\projlim</code>
$\varinjlim$	<code>\varinjlim</code>	$\varprojlim$	<code>\varprojlim</code>

Примеры:

$$\overline{\lim}_{n \rightarrow \infty} a_n = \inf_n \sup_{m \geq n} a_m \quad \begin{array}{l} \text{\texttt{\$}\varlimsup_{\{n\to\infty\}}\\ \text{\texttt{a\_n=\inf\_n\sup_{\{m\geq n\}}a\_m}} \end{array}$$

$$\mathcal{F}_x = \varinjlim_{U \ni x} \mathcal{F}(U) \quad \begin{array}{l} \text{\texttt{\$}\mathcal{F}_x=} \\ \text{\texttt{\varinjlim_{\{U\ni x\}}\mathcal{F}(U)}} \end{array}$$

(см. с. 55 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. 46), 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 ЛАТЭХ'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 `\iiiiint` 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):

$$\int_0^1 x^2 dx = 1/3 \quad \begin{array}{l} \text{\texttt{\$}\int_0^1 x^2 dx=1/3} \\ \text{\texttt{\$}} \end{array}$$

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:

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

$\begin{array}{l} \backslash[ \\ \backslashint\limits_0^1 x^2 dx=1/3 \\ \backslash] \end{array}$

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»:

$$\prod_{i=1}^n i = n!$$

$\begin{array}{l} \backslash[ \\ \backslashprod\nolimits_{i=1}^n i=n! \\ \backslash] \end{array}$

### 1.3. Разное

Мы уже перечислили почти все символы, используемые ЛАТ<sub>Э</sub>X'ом в математических формулах. 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.

$\partial$	<code>\partial</code>	$\triangle$	<code>\triangle</code>	$\angle$	<code>\angle</code>
$\infty$	<code>\infty</code>	$\forall$	<code>\forall</code>	$\exists$	<code>\exists</code>
$\emptyset$	<code>\emptyset</code>	$\neg$	<code>\neg</code>	$\aleph$	<code>\aleph</code>
$'$	<code>\prime</code>	$\hbar$	<code>\hbar</code>	$\nabla$	<code>\nabla</code>
$\imath$	<code>\imath</code>	$\jmath$	<code>\jmath</code>	$\ell$	<code>\ell</code>
$\sqrt{\phantom{x}}$	<code>\surd</code>	$\flat$	<code>\flat</code>	$\sharp$	<code>\sharp</code>
$\natural$	<code>\natural</code>	$\top$	<code>\top</code>	$\bot$	<code>\bot</code>
$\wp$	<code>\wp</code>	$\Re$	<code>\Re</code>	$\Im$	<code>\Im</code>
$\backslash$	<code>\backslash</code>	$\parallel$	<code>\parallel</code>	$\spadesuit$	<code>\spadesuit</code>
$\clubsuit$	<code>\clubsuit</code>	$\diamondsuit$	<code>\diamondsuit</code>	$\heartsuit$	<code>\heartsuit</code>
$\dagger$	<code>\dag</code>	$\S$	<code>\S</code>	$\copyright$	<code>\copyright</code>
$\ddagger$	<code>\ddag</code>	$\P$	<code>\P</code>	$\pounds$	<code>\pounds</code>

Последние шесть символов (от  $\dagger$  до  $\pounds$ ) можно использовать не только в формулах, но и в тексте.

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  $\parallel$ , 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; именно из-за этой равносильности запись  $x'^2$  приводит к ошибке (см. с. 25).

Про обыкновенные символы создатели пакета `amssymb` тоже не забыли. При его подключении открывается доступ к следующим:

$\square$	<code>\square</code>	$\blacksquare$	<code>\blacksquare</code>
$\diamond$	<code>\lozenge</code>	$\blacklozenge$	<code>\blacklozenge</code>
$\backprime$	<code>\backprime</code>	$\bigstar$	<code>\bigstar</code>
$\blacktriangledown$	<code>\blacktriangledown</code>	$\blacktriangle$	<code>\blacktriangle</code>
$\triangledown$	<code>\triangledown</code>	$\angle$	<code>\angle</code>
$\measuredangle$	<code>\measuredangle</code>	$\sphericalangle$	<code>\sphericalangle</code>
$\textcircled{S}$	<code>\circledS</code>	$\complement$	<code>\complement</code>
$\diagup$	<code>\diagup</code>	$\diagdown$	<code>\diagdown</code>
$\varnothing$	<code>\varnothing</code>	$\nexists$	<code>\nexists</code>

$\Finv$	<code>\Finv</code>	$\Game$	<code>\Game</code>
$\mho$	<code>\mho</code>	$\eth$	<code>\eth</code>
$\beth$	<code>\beth</code>	$\gimel$	<code>\gimel</code>
$\daleth$	<code>\daleth</code>	$\digamma$	<code>\digamma</code>
$\varkappa$	<code>\varkappa</code>	$\Bbbk$	<code>\Bbbk</code>
$\hslash$	<code>\hslash</code>	$\hbar$	<code>\hbar</code>

Из этого набора нам уже знакомы греческая буква  $\varkappa$  и обозначение для пустого множества  $\emptyset$ .

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

$*$	<code>*</code>	или	<code>\ast</code>
$\neq$	<code>\ne</code>	или	<code>\neq</code>
$\leq$	<code>\le</code>	или	<code>\leq</code>
$\geq$	<code>\ge</code>	или	<code>\geq</code>
$[$	<code>[</code>	или	<code>\lbrack</code>
$]$	<code>]</code>	или	<code>\rbrack</code>
$\{$	<code>\{</code>	или	<code>\lbrace</code>
$\}$	<code>\}</code>	или	<code>\rbrace</code>
$\rightarrow$	<code>\to</code>	или	<code>\rightarrow</code>
$\leftarrow$	<code>\gets</code>	или	<code>\leftarrow</code>
$\ni$	<code>\ni</code>	или	<code>\owns</code>
$\wedge$	<code>\wedge</code>	или	<code>\land</code>
$\vee$	<code>\vee</code>	или	<code>\lor</code>
$\neg$	<code>\neg</code>	или	<code>\lnot</code>
$\Vdash$	<code>\Vdash</code>	или	<code>\Vdash</code>

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

$\dashrightarrow$	<code>\dashrightarrow</code>	или	<code>\dashrightarrow</code>
$\doteq$	<code>\Doteq</code>	или	<code>\doteqdot</code>
$\cup$	<code>\Cup</code>	или	<code>\doublecup</code>
$\cap$	<code>\Cap</code>	или	<code>\doublecap</code>
$\ll$	<code>\lll</code>	или	<code>\llless</code>
$\gg$	<code>\ggg</code>	или	<code>\gggtr</code>

## 2. Важные мелочи

### 2.1. Нумерация формул

В математических текстах обычно приходится для удобства ссылок нумеровать формулы;  $\text{\LaTeX}$  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 `\[` или `$$` быть не должно!). Каждая такая формула на печати автоматически получит номер. Чтобы на него можно было ссылаться (а зачем еще нумеровать?), 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. 20; напомним, что может понадобиться повторный запуск  $\text{\LaTeX}$ ). Поясним сказанное примером:

Как известно,

$$7 \times 9 = 63. \quad (1)$$

.....  
Из формулы (1) и того, что деление  
обратно к умножению, следует,  
что  $63/9 = 7$ .

Как известно,

```
\begin{equation}
\label{trivial}
7\times9=63.
\end{equation}
```

.....  
Из формулы~(\ref{trivial})  
и того, что деление  
обратно к умножению,  
следует, что  $63/9=7$ .

Знак ~ мы поставили, чтобы номер формулы и слово «формулы» не попали на разные строки (см. с. 91). 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. 13, 142): 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 командой `\eqref`, имеет вид (3.5).

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

Простое тождество

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

известно каждому школьнику.

Простое тождество

$$\begin{array}{l} \backslash[ \\ 7\backslashtimes 9=63\backslasheqno (3.2) \\ \backslash] \end{array}$$

известно каждому школьнику.

Номером выключной формулы, нумеруемая с помощью команды `\eqno`, будет служить весь текст, заключенный между `\eqno` и закрывающими формулу `\]` или `$$`; этот текст обрабатывается Т<sub>Е</sub>X'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`, Т<sub>Е</sub>X не создает, и в этом случае за корректность ссылок отвечаете только вы.

## 2.2. Переносы в формулах

При необходимости Т<sub>Е</sub>X может перенести часть внутритекстовой формулы на другую строку. Такие переносы возможны после знаков «бинарных отношений», наподобие знака равенства<sup>2</sup> (см. с. 41) или «бинарных операций», наподобие знаков сложения или умножения (см. с. 40), причем последний знак в строке, вопреки российской традиции, не дублируется в начале следующей. Чтобы избежать этих переносов, можно воспользоваться тем обстоятельством, что Т<sub>Е</sub>X не разрывает при переносе часть формулы, заключенную в фигурные скобки. Например, можно заключить в фигурные скобки всю формулу, в которой произошел нежелательный перенос, от открывающего ее знака доллара до закрывающего: тогда 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

```
\binoppenalty=10000
```

и/или строку

```
\relpenalty=10000
```

---

<sup>2</sup>Стрелки также рассматриваются Т<sub>Е</sub>X'ом как бинарные отношения.

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 несколько меньше, чем при заключении всей формулы в фигурные скобки.

Для любознательных поясним, что `\binoppenalty` и `\relpenalty` — параметры (Т<sub>Е</sub>X'овские), значением которых может быть целое число. 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 большие значения, тогда вероятность разрывов уменьшится. Значение 10000 означает абсолютный запрет.

При заключении всей формулы в фигурные скобки верстка абзацев затрудняется, поскольку Т<sub>Е</sub>X лишается возможности варьировать в ней интервалы между символами для выравнивания строк.

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 будет 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, Т<sub>Е</sub>X не переносит никогда. 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 **P**, набранная прямым жирным шрифтом. Тогда надо воспользоваться командой `\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 VI.

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

$\mathbf{x+y}$	<code>\mathbf x+y\$</code>
$\mathrm{x+y}$	<code>\mathrm x+y\$</code>
$\mathtt{x+y}$	<code>\mathtt x+y\$</code>
$\mathsf{x+y}$	<code>\mathsf x+y\$</code>
$\mathcal{T}_X$	<code>\mathcal T_X\$</code>
$\Gamma+y$	<code>\mathit\Gamma+y\$</code>

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

При подключении пакета `amsmath` возникает возможность набирать буквы в формулах полужирным курсивом. Для этого нужно воспользоваться командой `\boldsymbol`: чтобы на печати вышло  $\boldsymbol{x+y}$ , в исходном тексте надо набрать `\boldsymbol x+y$`.

С другой стороны, 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 надо использовать команды наподобие `\varGamma`, о которых шла речь в разд. II.1.1.

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  $X_{\mathrm{sing}}$ .

Множество особенностей многообразия  $\mathbf{X}$  обозначается  $\mathbf{X}_{\mathrm{sing}}$ .

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. 16: 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. 16 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. 95.

Если подключить стилевой пакет `amssymb`, то в математических формулах можно использовать еще два шрифта: ажурный ( $\mathbb{R}, \mathbb{C}, \mathbb{Q}, \dots$ ) и готический ( $\mathfrak{S}, \mathfrak{p}, \mathfrak{g}, \dots$ ). 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.

Теперь, когда вы знаете, как печатать символы в формулах прямым шрифтом, может возникнуть искушение восполнить отсутствие в стандартном комплекте L<sup>A</sup>T<sub>E</sub>X'a команды, дающей функцию `tg`, путем набора чего-нибудь вроде `\mathrm{tg}x`. Так делать, однако, не надо, поскольку при этом пробелы будут неправильными:

`tgx`, но `\sin x`

`\mathrm{tg} x`,  
но `\sin x`

Правильно действовать так, как рекомендуется на с. 46. Если вам хочется узнать, почему все так получается, прочтите разд. 5.4.

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

$\sqrt{x^3} = x$  for all  $x$ .  
`\[`  
`\sqrt{x^3}=x \mathrm{for all} x`.  
`\]`

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

Отметим, 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 L<sup>A</sup>T<sub>E</sub>X'e пока сохраняется (ради совместимости со старыми версиями) еще один способ, с которым можно ознакомиться из приведенной ниже таблицы.

Правильный способ:

`\mathrm x+y`  
`\mathbf x+y`  
`\mathsf x+y`

Устаревший способ:

`\rm x)+y`  
`\bf x)+y`  
`\sf x)+y`

Получается:

$x + y$   
 $\mathbf{x} + y$   
 $\mathsf{x} + y$

<code>\mathtt x+y</code>	<code>{\tt x}+y</code>	$x + y$
<code>\mathcal T_X</code>	<code>{\cal T}_X</code>	$\mathcal{T}_X$

## 2.4. Включение текста в формулы

Если у вас подключен стилевой пакет `amsmath`, то включить фрагмент обычного текста в математическую формулу можно с помощью команды `\text`. В следующем примере продемонстрировано, как это можно сделать; в нем используется еще команда `\qquad`, делающая в тексте или формуле пробел размером `2em` (см. с. 19 по поводу единиц длины, применяемых Т<sub>Е</sub>X’ом); подробнее по поводу команд, создающих пробелы в формулах, см. разд. 5.1; по поводу команд, создающих пробелы в тексте, см. разд. III.3.2.

$$\sqrt{x^3} = x \quad \text{для всех } x.$$

```

\[
\sqrt{x^3}=x\qquad
\text{для всех } x.
\]
```

Аргумент команды `\text` обрабатывается Т<sub>Е</sub>X’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 `\text` дать команду смены шрифта в тексте). Весь текст, являющийся аргументом команды `\text`, 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 `\text`, рассматривается Т<sub>Е</sub>X’ом как одна большая буква; пробел в формуле между «буквой», содержащей текст из `\text`, и буквой  $x$  будет недостаточен). Команда `\qquad` была использована по аналогичной причине.

На самом деле можно было бы написать даже так:

```

\[
\sqrt{x^3}=x\qquad\text{для всех } x$.}
\]
```

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

Текст, включенный в формулы с помощью команды `\text`, будет правильно менять размеры в степенях и индексах, числителе и знаменателе дробей, набранных с помощью `\frac`, и других подобных контекстах.

Если пакет `amsmath` по какой-то причине не подключен, то вместо `\text` можно использовать L<sup>A</sup>T<sub>E</sub>X'овскую команду `\mbox`: она будет работать так же, как `\text`, но при этом в тексте, включенном в формулу с помощью `\mbox`, размер шрифта не изменится, в какую бы часть формулы этот текст ни попал.

Команду `\text` можно использовать и вне математических формул.

Хотя команда `\mbox` не очень удобна для включения текста в формулы, она очень важна во многих других ситуациях; мы с ней еще не раз встретимся.

## 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 T<sub>E</sub>X'e на этот случай предусмотрен механизм автоматического выбора размера скобок. Пользуются им так.

В формуле

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

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. T<sub>E</sub>X'нический термин для таких символов — «ограничители» (по-английски *delimiters*).

(	(	)	)	[	[
]	]	{	\{	}	\}
[	\lfloor	]	\rfloor	[	\lceil
]	\rceil	<	\angle	>	\rangle
			\	/	/
\	\backslash				

Вместо `\left\langle` можно писать `\left<`, и аналогичным образом вместо `\right\rangle` можно писать `\right>` (однако же `<` нельзя писать вместо `\langle`: без `\left` и `\right` это разные символы). Кроме знаков, перечисленных в этой таблице, менять свои размеры под действием `\left` и `\right` могут и вертикальные стрелки из таблицы на с. 44.

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

`\ulcorner`   `\urcorner`   `\llcorner`   `\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):

$$M(f) = \left( \int_a^b f(x) dx \right) / (b - a)$$

```

\[\begin{aligned}
M(f)=\left.\left(\right.\right. & \\
& \int\limits_a^bf(x)\,dx \\
& \left.\right) \\
& \left./\right(b-a) \\
& \]
\end{aligned}
```

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:

$$\int_a^b \frac{1}{2}(1+x)^{-3/2} dx = -\frac{1}{\sqrt{1+x}} \Big|_a^b$$

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

Здесь, кстати, мы не поставили  $\backslash$ , 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. Вот пример:

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

Для удобочитаемости хотелось бы, чтобы внешние знаки модуля были выше внутренних, но поскольку в формуле выступающих элементов нет, то и команды `\left` и `\right` не считают нужным увеличить ограничители, в которые формула заключена.

Автоматически получающиеся ограничители бывают и слишком велики, как в следующем примере:

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

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

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

$$||x+1|-|x-1|| \qquad \$\bigr| \, |x+1|-|x-1| \bigr| \$$$

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

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

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

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.

Если пакет `amsmath` не подключен, а «основной шрифт» документа крупнее, чем кегль 10 (т. е. если указаны классовые опции `11pt` или `12pt` — см. с. 142 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».

## 2.6. Перечеркнутые символы

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

Множество $\{x \mid x \not\in x\}$ существо-	Множество $\{x \mid x \not\in x\}$
вать не может. В этом состоит	существовать не может. В этом
парадокс Рассела.	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 лучше вместо команды `\not` пользоваться готовыми командами для перечеркнутых символов (с. 43).

## 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 `amssymb`.

## 2.8. Надстрочные знаки

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

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

$$\overline{a_n a_{n-1} \dots a_1 a_0} = 10^n a_n + \dots + a_0.$$

$$\begin{array}{l} \backslash[ \\ \overline{a_{na_{n-1}}\ldots} \\ a_1a_0]=10^na_n+\ldots+a_0. \\ \backslash] \end{array}$$

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

<code>\hat a</code>	$\hat{a}$	<code>\check a</code>	$\check{a}$
<code>\tilde a</code>	$\tilde{a}$	<code>\acute a</code>	$\acute{a}$
<code>\grave a</code>	$\grave{a}$	<code>\dot a</code>	$\dot{a}$
<code>\ddot a</code>	$\ddot{a}$	<code>\breve a</code>	$\breve{a}$
<code>\bar a</code>	$\bar{a}$	<code>\vec a</code>	$\vec{a}$

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 *i* и *j* (см. таблицу на с. 49):

Лучше $\tilde{i}$ , чем $\tilde{i}$ .	Лучше <code>\$\tilde{\imath}</code> , чем <code>\$\tilde{i}</code> .
---------------------------------------	---

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

Тождество $\widehat{f * g} = \hat{f} \cdot \hat{g}$	Тождество <code>\$\widehat{f*g}=\hat{f}\cdot\hat{g}</code> означает,
означает, что преобразование Фурье переводит свертку в произведение.	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  $\hat{\hat Z}$ , Правильно  $\hat{\hat Z}$ , а не  $\hat{\hat Z}$ .  
 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.

$\TeX$  позволяет ставить надстрочные знаки над буквами не только в математической формуле, но и в обычном тексте (такие знаки называются «диакритическими»), но команды для постановки этих знаков совершенно другие. Об этом — на с. 94.

### 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 скромных средствах набора матриц, доступных в «чистом»  $\LaTeX$ ’е.

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

	<code>\[</code>	<code>\begin{pmatrix}</code>
$\begin{pmatrix} a_{11} - \lambda & a_{12} & a_{13} \\ a_{21} & a_{22} - \lambda & a_{23} \\ a_{31} & a_{32} & a_{33} - \lambda \end{pmatrix}$	<code>a_{11}-\lambda &amp; a_{12}&amp;a_{13}\\</code>	<code>\end{pmatrix}</code>
	<code>\]</code>	

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  $\text{\TeX}$ -овского файла; в одной строке  $\text{\TeX}$ -овского файла можно поместить текст, соответствующий на печати нескольким строкам матрицы. Короче говоря, в окружении `pmatrix` также действует  $\text{\TeX}$ -овский принцип «конец строки равен 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  $\text{\TeX}$ -ническом языке это действие называется «присваивание нового значения счетчику `MaxMatrixCols`» — см. гл. VI). 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:

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

Исходный текст для него выглядит так:

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



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:

$[X, Y] = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	<code>\$[X,Y]=\bigl(\begin{smallmatrix}</code>
	<code>1 &amp; 0\\0 &amp; -1</code>
	<code>\end{smallmatrix}\bigr)\$</code>

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  $\text{\LaTeX}$ овским окружением `array`. Вот как получить этими средствами пример со с. 63:

$\left( \begin{array}{ccc} a_{11} - \lambda & a_{12} & a_{13} \\ a_{21} & a_{22} - \lambda & a_{23} \\ a_{31} & a_{32} & a_{33} - \lambda \end{array} \right)$	<code>\[\left(\begin{array}{ccc}</code>
	<code>a_{11}-\lambda &amp; a_{12}&amp;a_{13}\\</code>
	<code>a_{21}&amp; a_{22}-\lambda &amp;a_{23}\\</code>
	<code>a_{31}&amp; a_{32}&amp;a_{33}-\lambda</code>
	<code>\end{array}\right)</code>
	<code>\]</code>

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»  $\text{\LaTeX}$ 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:

$$\begin{array}{ll} \frac{2}{3} \text{ и } \frac{2}{3} & \$\frac{2}{3}$ и $\dfrac{2}{3}$ \\ 2^{\frac{3}{5}} \text{ и } 2^{\frac{3}{5}} & $2^{\{\frac{3}{5}\}}$ и $2^{\{\tfrac{3}{5}\}}$ \end{array}$$

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  $\Gamma_{ij}^k$  писали  $\left\{ \begin{smallmatrix} ij \\ k \end{smallmatrix} \right\}$ .

Раньше вместо  $\sim \Gamma_{ij}^k$  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 VI, 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. 58); 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  $\left( \frac{x}{y-z} \right)$  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 использовании командой `\dfrac` (в разд. 5.2 вы узнаете, что в  $\TeX$ ’нической терминологии это называется *displaystyle*), цифра 1 — размер, как при использовании 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 Т<sub>Е</sub>X'овской командой `\atop`:

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

Раньше вместо  $\Gamma_{ij}^k$  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 Т<sub>Е</sub>X'овская команда `\choose`:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \begin{array}{l} \backslash[ \\ \{n\choose k}=\frac{n!}{k!(n-k)!} \\ \backslash] \end{array}$$

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»:

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

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:

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

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

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 L<sup>A</sup>T<sub>E</sub>X'овская команда `\stackrel`. У этой команды два аргумента: первый — то, что будет над строкой, второй — то, что останется в строке:

$$A \xrightarrow{f} B$$

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

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:

$$A \stackrel{f}{\leftarrow} B \stackrel{f+g-h}{\rightarrow} C \quad \begin{array}{l} \backslash [ A \backslash xleftarrow [z] {f} B \\ \backslash xrightarrow {f+g-h} C \backslash ] \end{array}$$

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, если у вас подключен пакет `amsmath`, то формула

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

получается следующим образом:

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

Если `amsmath` не подключен, придется воспользоваться командой `\mbox` вместо `\text`.

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:

$$\underbrace{\overbrace{a+b+\ldots+z}_{\text{\texttt{\texttt{a+b+\ldots+z}}_{26}+1+\ldots+10}}^{36}}_{26} \quad \begin{array}{l} \backslash [ \\ \backslash overbrace {\backslash underbrace { \\ a+b+\backslash dots+z \\ }_{\texttt{\texttt{26}}+1+ \\ \backslash dots+10}}^{\texttt{\texttt{36}}} \\ \backslash ] \end{array}$$

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. 84).



## 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. Если оформить каждую из этих строк как отдельную выключную формулу с помощью `\[...\]` или `$$...$$` и записать эти выключные формулы подряд, то 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 то небольшое, чего можно добиться без подключения дополнительных пакетов.

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

$1 + 2 + 3 + 4 + \dots$	<code>\begin{multline}</code>
$+ 46 + 47 + 48 + \dots$	<code>1+2+3+4+\ldots\\</code>
$+ 99 + 100 = 5050$	<code>+46+47+48+\ldots\\</code>
(2)	<code>+99+100=5050</code>
	<code>\end{multline}</code>

Первая из строк печатается выключенной влево, последняя — выключенной вправо, остальные строки центрируются. Окружение `multline`, как и окружение `equation`, не должно быть заключено в знаки `\[` и `\]` или в пары долларов. 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. 18).

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 `\[` и `\]` или окружения `equation`, но воспользоваться окружением `gather`:

$$\begin{array}{ll} 2 \times 2 = 4 & (3) \\ 9 \times 9 = 81 & (4) \end{array}$$

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

При использовании `gather` формулы также не должны быть заключены в символы `\[` и `\]` (или `$$`). Каждая из формул, собранных в `gather`, автоматически нумеруется. Чтобы на пронумерованную таким образом формулу можно было сослаться, надо ее пометить, поставив перед `\` команду `\label` (см. примеры меток и ссылок в разд. 2.1; подробности — в разд. IV.10 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`:

$$\begin{array}{l} 1999 = 1000 + 900 + \\ \quad + 90 + 9 \end{array}$$

```
\[
\begin{split}
1999&=1000+900+{\}\\
&\quad +90+9
\end{split}
\]
```

Разбиение формулы на строки по-прежнему задается с помощью `\`, а знак `&` стоит перед символами, по которым производится выравнивание. По Т<sub>Е</sub>Хническим причинам формулу, разбитую на строки помощью `split`, нельзя задавать с помощью знаков `$$` (окружением `equation` пользоваться можно).

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`:

$$\begin{array}{rcl}
 7 \times 9 = 63 & 63 : 9 = 7 & (5) \\
 9 \times 10 = 90 & 90 : 10 = 9 & (6)
 \end{array}$$

```

\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 `[`, `\]` или `$$`, 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 ключевые формулы содержат текстовые комментарии. Желательно, чтобы эти комментарии были выровнены. Вот как можно этого добиться с помощью `align`:

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

```

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

```

Обратите внимание на два амперсанда, отделяющие комментарий от формул (см. выше текст мелким шрифтом). Нелишние 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:

$$\begin{array}{rcl}
 3 \cdot 5 + 7 \cdot 5 = (3 + 7) \cdot 5 & \text{(ясно)} & \\
 = 50 & \text{(очевидно)}, & \\
 \text{откуда} & & \\
 15 + 35 = 50 & &
 \end{array}$$

```

\begin{align*}
3\cdot 5+7\cdot 5&=(3+7)\\
&\cdot 5 \&\&\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:

$$\begin{cases} x^2 + y^2 = 7 \\ x + y = 3. \end{cases}$$

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

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).

Существует также окружение `gathered`, которое относится к `gather` так же, как `aligned` относится к `align`: оно создает столбик из нескольких выровненных по центру формул, который можно использовать в качестве составной части большей формулы (и также нельзя использовать самостоятельно, без знаков `\[` или `$$`).

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:

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

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

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. Мы приводим эти сведения не для того, чтоб читатель использовал их при самостоятельном наборе — напротив, пользоваться этими средствами мы не советуем — но в справочных целях, на случай, если читатель столкнется с готовым набором, в котором обходятся средства из пакета `amsmath`.

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

$$\begin{cases} x^2 + y^2 = 7 \\ x + y = 3. \end{cases}$$

```

\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`), и бороться с этим трудно; проще подключать 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:

$$\begin{array}{rcl} 2 \times 3 & = & 6 \\ 2 + 3 & = & 5 \end{array} \quad \begin{array}{l} (7) \\ (8) \end{array}$$

```

\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. 91); для аналогичных целей мы использовали этот символ и вторично.

При использовании окружения `eqnarray` не надо писать знаки `\left` или `$$` (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 `\\`):

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

$$\sqrt{576} = 24 \quad (9)$$

```
\begin{eqnarray}
\int_{-\infty}^{\infty}
e^{-x^2}dx &= & \sqrt{\pi}
\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*`:

$$x^{20} = (x-1)^{20} + 20(x-1)^{19} + \dots + 20(x-1) + 1$$

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

перед первым знаком + во второй строке формулы мы поставили пару из открывающей и закрывающей фигурных скобок, чтобы на печати знак + не подошел слишком близко к первому символу второй строки. Природа описанного эффекта объясняется ниже в разд. 5.

### 4.3. Простейшие коммутативные диаграммы

Для набора в  $\text{\LaTeX}$ е «коммутативных диаграмм» без наклонных, изогнутых или пунктирных стрелок можно использовать стилевой пакет `amscd`. Если этот пакет подключен, то коммутативная диаграмма оформляется в виде окружения `CD`. Рассмотрим, например, диаграмму

$$\begin{array}{ccccccc} 0 & \longrightarrow & E' & \xrightarrow{f} & E & \xrightarrow{g} & E'' \longrightarrow 0 \\ & & \downarrow p & & \downarrow q & & \downarrow r \\ 0 & \longrightarrow & F' & \xrightarrow{f} & F & \xrightarrow{g} & F'' \longrightarrow 0 \end{array}$$

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  $\text{\LaTeX}$  не сбился со счета, выясняя, в какие колонки ставить вертикальные стрелки.

Опишем работу окружения  $\text{\CD}$  более аккуратно. Каждую коммутативную диаграмму окружение  $\text{\CD}$  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  $\text{\TeX}$  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»:

$$\begin{array}{ccccc}
 A & \xlongequal{\quad} & B & \longrightarrow & C \\
 v_1 \downarrow & & f \uparrow & & \parallel \\
 D & \xleftarrow{\quad g \quad} & E & \longleftarrow & F
 \end{array}$$

```

\[\begin{CD}
A @= B @>>> C \\
@V{V_1}VV @A{f}AA @| \\
D @<<g< E @<<< F
\end{CD}\]

```

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.

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)  $\text{\LaTeX}$ 'е набор диаграмм не предусмотрен. На самый крайний случай, если нет ни `amscd`, ни `Xy-pic`'а, можно сделать так:

```

\[\begin{array}{c}
\backslash\begin{array}{c}
0&\backslash\longrightarrow&E'&\\
\stackrel{f}{\backslash\longrightarrow}&E&\\
\stackrel{g}{\backslash\longrightarrow}&\\
E''&\backslash\longrightarrow&0\\
&\&\downarrow\lefteqn{p}&\&\downarrow\\
\lefteqn{q}&\&\downarrow\lefteqn{r}&\\
0&\backslash\longrightarrow&F'&\\
\stackrel{f}{\backslash\longrightarrow}&F&\\
\stackrel{g}{\backslash\longrightarrow}&F''&\\
&\backslash\longrightarrow&0
\end{array}
\end{array}\]

```

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  $\text{\TeX}$  учитывает, сколько места занимает тот или иной фрагмент формулы. В  $\text{\TeX}$ 'е



предусмотрены специальные команды, позволяющие фальсифицировать эти данные. В частности, команда `\lefteqn` печатает формулу, являющуюся ее аргументом, но при этом сообщает Т<sub>Э</sub>X’у, что по горизонтали эта формула не занимает места вообще. Стало быть, с точки зрения Т<sub>Э</sub>X’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 Т<sub>Э</sub>X’а Дональд Кнут назвал такого рода приемы работы с Т<sub>Э</sub>X’ом «грязными трюками» (dirty tricks). Впрочем, при написании Т<sub>Э</sub>X’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 двумя важными исключениями: во-первых, часто бывают нужны диаграммы с наклонными или изогнутыми стрелками (о них мы рассказываем в приложении Д), и во-вторых, при написании математической статьи полезно знать, как грамотно оформлять тексты теорем, определений и тому подобные вещи. Об таком оформлении у нас рассказывается в разд. 3 и 3.1 главы VI; вы можете прочитать это уже сейчас, пропуская непонятные места и справляясь при необходимости с разд. IV.5. Если вам все же не хватает символов для формул, то см. с. 91 по поводу того, где еще их взять.

### 5. Тонкая настройка

В этом разделе мы рассмотрим некоторые более изысканные вопросы, связанные с набором математических формул. Мелкий шрифт при первом чтении лучше пропустить.

#### 5.1. Пробелы вручную

Бывают случаи, когда промежутки между символами в формулах, выбранные Т<sub>Э</sub>X’ом автоматически, выглядят неудачно. В этом случае в формулу можно включить команды, задающие промежутки в явном виде. Вот основные из них:

<code>\quad</code>	Пробел в 1em:
<code>\qqquad</code>	Пробел в 2em:
<code>\,</code>	«Тонкий пробел», или тонкая шпация:
<code>\:</code>	«Средний пробел»:
<code>\;</code>	«Толстый пробел»:
<code>\!</code>	«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  $\int f(x) dx$ ,  $\iint f dx dy$  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 `\quad` полезна для отделения текста, входящего в формулу, от собственно формулы (см. с. 57). 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»:

<code>displaystyle</code>	«выключной» стиль
<code>textstyle</code>	«текстовый» стиль
<code>scriptstyle</code>	стиль для индексов
<code>scriptscriptstyle</code>	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 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 `scriptscriptstyle` 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: 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. 70 показано, какой неудачный результат выйдет, если набирать цепную дробь наивным образом: 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 почему-то 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. 80 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, Т<sub>Е</sub>X 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 тоже ни к чему). Такой невидимый символ генерируется Т<sub>Е</sub>X'овской командой `\mathstrut`:

В формуле  $\sqrt{a} + \sqrt{d}$  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 Т<sub>Е</sub>X'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 Т<sub>Е</sub>X'а, а затем аккуратно стерта с бумаги. Пример:

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

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 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 Т<sub>Е</sub>X'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`:

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

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

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  $\text{\TeX}$  считает, что места по горизонтали она не занимает. Поэтому формула  $1 + \overbrace{2 + 3} + 4$  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. 80).

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  $\text{\TeX}$  при расстановке интервалов в математических формулах. В стандартных ситуациях мы об этом не задумываемся, а полностью доверяем  $\text{\TeX}$ ’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. 81) и при этом не хотим подбирать верные интервалы экспериментально.

При наборе формулы  $\text{\TeX}$  рассматривает ее как состоящую из частей одного из следующих типов:

Обыкновенный символ	например, <code>\alpha</code>
Бинарная операция	см. с. 40
бинарное отношение	см. с. 41
Математический оператор	см. с. 45, 48
Подформула	например, <code>{x^2}</code>
Знак препинания	, или ; или <code>\colon</code> или <code>\ldotp</code>
Скобка	

Здесь подформула — это любой фрагмент формулы, заключенный в фигурные скобки. Команда `\colon` задает двоеточие, рассматриваемое как знак препинания (двоеточие, набранное непосредственно, рассматривается  $\text{\TeX}$ ’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 Т<sub>Е</sub>X'a) относятся также все стрелки (с. 44) и фрагменты формул, создаваемые командой `\stackrel`. При расстановке пробелов в формуле Т<sub>Е</sub>X 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.<sup>3</sup> Подформула (т. е. фрагмент формулы, заключенный в фигурные скобки) рассматривается Т<sub>Е</sub>X'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. 24, когда обсуждали, как задать в Т<sub>Е</sub>X'е десятичную дробь. Мы не будем вдаваться в точные правила расстановки пробелов (они перечислены в книге [2]). Для нас сейчас важнее то, что Т<sub>Е</sub>X 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 Т<sub>Е</sub>X рассматривал  $\hat{\otimes}$  не как обычный символ, а как символ бинарной операции, надо сделать так:

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

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

---

<sup>3</sup>Иногда полезно поставить пустые фигурные скобки {}, чтобы создать фиктивный аргумент бинарной операции и тем самым обеспечить желательные пробелы; мы делали это в примере на с. 73.

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. VI по поводу того, как это сделать).

Типичный пример использования команды `\mathop` — определение имени операции, записываемой прямым шрифтом (см. с. 46, где мы давали определение функции `tg`). Обозначения такого типа встречаются в математических текстах очень часто, и набора команд для них, предусмотренного L<sup>A</sup>T<sub>E</sub>X'ом (с. 45), вполне может не хватить; в этом случае, чтобы получить на печати, скажем,  $\text{Ext}^1(E, F)$ , можно было бы написать

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

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

Сравните  $\sin x$  и  $\sin x$ .

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. 49). Именно таким образом работает определенная в пакете `amsmath` команда `\DeclareMathOperator`.

А вот пример, когда T<sub>E</sub>X'у надо объяснить, что некоторый сложный символ есть символ математического оператора. Пусть нам понадобилась формула

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

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

$$\sum'_{x \in \Gamma} f(x) \qquad \begin{array}{l} \backslash[ \\ \backslash\text{sum}'_{x \in \Gamma} f(x) . \\ \backslash] \end{array}$$

В самом деле, из сказанного на с. 50 вытекает, что наша запись равносильна такой:

$$\begin{array}{l} \backslash[ \\ \backslash\text{sum}^{\prime}_{x \in \Gamma} f(x) . \\ \backslash] \end{array}$$

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

$$\begin{array}{l} \backslash[ \\ \backslash\mathop{\{\sum\}}'_{x \in \Gamma} f(x) . \\ \backslash] \end{array}$$

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 Т<sub>E</sub>X'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. 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<sup>4</sup>. Для этих целей в Т<sub>E</sub>X'е предусмотрен параметр `\mathsurround` (см. разд. I.2.9 по поводу Т<sub>E</sub>X'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 Л<sup>A</sup>T<sub>E</sub>X'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`).

---

<sup>4</sup>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

Во всех системах, основанных на Т<sub>E</sub>X'e, существуют дефис - (по-английски hyphen), короткое тире – (en-dash) и длинное тире — (em-dash). Знак минуса —, отличающийся от обоих тире, встречается только в математических формулах, и там он, как вы помните, изображается просто знаком - (см. разд. I.3).

Чтобы получить на печати дефис, короткое тире или длинное тире, надо в исходном тексте набрать один, два или три знака - соответственно.

Любознательный читатель может спросить, почему запись **жж** в исходном тексте дает на печати всего-навсего две буквы «ж», а запись **--** дает тире, которое шире, чем два дефиса. Ответ: Т<sub>E</sub>X'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 **ХО** (полученное на печати естественным образом) и **ХО** (набранное со специальной командой, убирающей кернинг).

Все вышесказанное относилось к английскому языку. При наборе русских текстов можно в принципе использовать «английское длинное тире», задаваемое как ---, в качестве собственно тире, «английское короткое тире», задаваемое двумя дефисами — для временных промежутков («2–3 часа», набираемое как *через 2--3 часа*), а набираемый непосредственно дефис для изображения дефиса — в надежде, что «чистовую отделку» текста проведет технический редактор.

Если подходить к набору текста с большим ригоризмом, то надо отметить, что принятые в русской полиграфии размеры тире отличаются от тех, что получаются при пользовании сочетаниями знаков --- или --. Чтобы получилось тире, больше соответствующее отечественной традиции, надо действовать следующим образом. Напомним, что при наборе русского текста в вашем файле, сразу после команды \documentclass, должны присутствовать следующие две:

```
\usepackage[кодировка]{inputenc}
\usepackage[russian]{babel}
```

где вместо *кодировка* следует написать cp1251, koi8-r, cp866 или utf8, в зависимости от того, какую из кириллических кодировок вы используете (см. с. 29, где приведен простейший пример ЛАТЭХ'овского файла, использующего русские буквы; в приложении И.5 рассказано о другом корректном способе оформления русских текстов в ЛАТЭХ'е). Если ваш файл начинается именно так, то для набора русского длинного тире следует воспользоваться сочетанием знаков "---- (для любознательных: это не лигатура).

## 1.2. Кавычки

В английских текстах открывающая кавычка изображается во входном тексте двумя подряд идущими обратными апострофами, закрывающая — двумя апострофами.

The “definitions” are The ‘‘definitions’’ are translations  
translations rather than rather than explanations.  
explanations.

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

В русских текстах употребляются кавычки типа «елочки» и „лапки“.

Открывающие и закрывающие «елочки» можно задавать либо сочетаниями << и >> соответственно (такие лигатуры присутствуют в русских шрифтах, используемых ЛАТЭХ'ом по умолчанию), либо сочетаниями "< и "> (этот способ призван сработать, даже если исходные русские шрифты заменить другими).

Кавычки-„лапки“ следуют задавать сочетаниями символов "‘ и ’" соответственно. Разумеется, все эти способы набора «русских» кавычек сработают только если в вашем файле после `\documentclass` присутствует вызов пакета `babel` с опцией  `russian`, как было сказано выше.

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 ЛАТЭХ’е этой цели служит команда `\, .` Пример:

«„Карова“ или „корова“, как правильно?» — спросил он.	<code>&lt;&lt;\, "‘Карова"’ или "‘корова"', как правильно?&gt;&gt;~"--- спросил он.</code>
---	--

Мы поставили символ `~` после закрывающих кавычек, чтобы тире заведомо напечаталось на той же строчке, что и предшествующее слово (см. с. 91 ниже).

### 1.3. Многоточие

При ЛАТЭХ’овском наборе для многоточия лучше не ставить три точки, а пользоваться специальной командой `\ldots` или `\dots`.

Вместо “...” пишем: Нет, что-то здесь не так...	Вместо “‘...’” пишем: Нет, что-то здесь не так <code>\ldots</code>
---	---

(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`.)

### 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` (для чего надо сказать в преамбуле `\usepackage{textcomp}`) — и тогда знак номера можно будет набирать с помощью команды `\textnumero`. (Заодно подключение пакета `textcomp` может несколько улучшить вид знака параграфа.) Помимо

знаков номера и параграфа, пакет `textcomp` открывает доступ к большому числу других типографских значков, с которыми можно ознакомиться в документации к пакету (см. начало гл. VIII по поводу того, где ее лучше всего искать в интернете). Впрочем, `textcomp` — отнюдь не единственный пакет, дающий доступ к новым символам. Хороший обзор таких пакетов (с символами как для текста, так и для формул) содержится во время от времени обновляемых файлах `symbols-a4.pdf` и `symbols-letter.pdf`. Искать их свежие версии удобно по запросу «The Comprehensive LaTeX Symbol List».

Наконец, в тексте можно использовать и любой из великого множества математических символов, если оформить его как математическую формулу:

Я ♡ TeX.

Я `\heartsuit` TeX.

## 2. Подчеркивания, рамки

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

Это слово будет подчеркнуто.

Это слово будет  
`\underline{подчеркнуто}`.

Подчеркнутый текст должен уместиться в одной строке.

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

Два слова будут в рамке.

Два слова будут `\fbox{в рамке.}`

Команда `\fbox` позволяет взять в рамку только фрагмент текста, уместающийся в одну строку. чтобы взять в рамку фрагмент, состоящий из нескольких строк, надо воспользоваться командами, о которых пойдет речь в гл. VII.

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  $\sim x$ . 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  $\sim$  после них с помощью текстового редактора) содержится в магическом слове «АВИКОСУЯ».

### 3.2. Установка промежутков вручную

Как команда «backslash с пробелом»  $\backslash$ , так и символ неразрывного пробела  $\sim$  генерируют пробел, но делать пробелы вручную с помощью набора чего-нибудь вроде  $\sim\sim\sim$  или  $\backslash\backslash\backslash$  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. 19). Для этого служат команды `\quad`, дающая промежуток в *1em*, и `\qqquad`, 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. 90.

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 <i>1em</i> промежутка.	Здесь <code>\quad 1em</code> промежутка. $\backslash\backslash$
Здесь <i>1em</i> промежутка.	Здесь <code>\quad\{1em</code> промежутка. $\backslash\backslash$
Здесь <i>1em</i> плюс еще немного.	Здесь <code>\quad\{ 1em</code> плюс еще немного.

В этом примере используется еще команда `\,`, начинающая новую строку (см. подробнее на с. 108).

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. 19), например, `\hspace{1.5cm}`

### 3.3. Для любознательных: промежутки между предложениями

Свойства  $\TeX$ 'а, описываемые ниже, связаны с особенностями именно английской полиграфии: команда `\usepackage[russian]{babel}` их отключает, так что в нормальном случае при наборе русского текста вы с ними не встретитесь. Последующий текст мелким шрифтом предназначен для читателей, хорошо владеющих английским языком.

В обычном режиме  $\TeX$  выравнивает справа строки абзаца, при необходимости делая переносы и слегка растягивая или сжимая промежутки между словами. При этом<sup>1</sup> промежутки между предложениями сами по себе шире и являются более растяжимыми, чем между словами внутри предложения (в следующем примере из «Винни-Пуха» все промежутки для наглядности равномерно растянуты):

North Pole. Discovered by Pooh. Pooh found it.

Точнее говоря, пробел увеличивается после следующих символов: точки, вопросительного знака, восклицательного знака, двоеточия (несколько меньше, чем после трех предыдущих), точки с запятой (еще меньше) и запятой (совсем чуть-чуть).

Впрочем, если последняя из букв, встретившихся перед одним из этих знаков препинания, была прописной, то пробел после этого знака препинания не увеличивается (поскольку точка после прописной буквы обычно обозначает не конец предложения, а конец чьих-то инициалов).

Как это обычно и бывает 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. 13).
- Если точка, вопросительный или восклицательный знак после прописной буквы заканчивает предложение, то перед ней следует поставить команду `\@` — тогда пробел будет обычным образом увеличен.

Вот примеры:

---

<sup>1</sup>Если этот эффект не отключен пакетом `babel` или каким-либо иным способом.

If  $n$  is even (resp. odd), then  $(-1)^n$  equals one (resp. negative one).

If  $n$  is even (resp. odd), then  $(-1)^n$  equals one (resp. negative one).

This research was supported by the NSF. The author is grateful to Prof. Smith.

This research was supported by the NSF. The author is grateful to Prof. Smith.

Если пробел задан как неразрывный с помощью символа `\~`, то он не увеличивается, невзирая ни на какие предшествующие знаки препинания.

## 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. Уже в исходном комплекте `TeX`'а имеются команды для набора таких букв из нескольких наиболее распространенных европейских языков. Команды для получения диакритических знаков собраны в нижеследующей таблице, где знаки проставлены, для примера, при букве «e».

Набрано	Вышло	Набрано	Вышло
<code>\'e</code>	è	<code>\'e</code>	é
<code>\^e</code>	ê	<code>\~e</code>	ẽ
<code>\=e</code>	ē	<code>\.e</code>	è
<code>\u{e}</code>	ě	<code>\v{e}</code>	ě
<code>\H{e}</code>	ě	<code>\"e</code>	ë
<code>\c{e}</code>	ç	<code>\d{e}</code>	ę
<code>\b{e}</code>	ë	<code>\r{e}</code>	ê
<code>\t oo</code>	ô		

В следующей таблице вы найдете команды для набора нескольких букв специального вида.

Набрано	Вышло	Набрано	Вышло
<code>\oe</code>	œ	<code>\OE</code>	Œ
<code>\ae</code>	æ	<code>\AE</code>	Æ
<code>\aa</code>	å	<code>\AA</code>	Å
<code>\o</code>	ø	<code>\O</code>	Ø
<code>\l</code>	ł	<code>\L</code>	Ł
<code>\i</code>	ı	<code>\j</code>	Ј
<code>\ss</code>	ß		

Команды `\i` и `\j` в этой таблице нужны, чтобы ставить диакритические знаки над буквами `i` и `j`: если просто сказать, допустим, `\=i`, то получится `ī`, а это не то, что требуется. Правильно писать `\=\i`. Вот несколько примеров.

Chérie, ça ne me plaît pas!	<code>Ch\'erie, \c{c}a ne me pla\^i t pas!</code>
Götterdämmerung	<code>G\'otterd\'ammerung</code>

Для набора обычных русских текстов из всего этого великолепия нужна только команда `\'` (чтобы ставить ударения). Буква `ё` в стандартных русских шрифтах присутствует, так что набирать ее как `\"е` необходимости обычно не возникает.

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.

Хотя перечисленные выше команды дают возможность набирать все знаки алфавита многих европейских языков, это еще не значит, что вы так просто сможете набирать длинные тексты на этих языках: если просто так вставить в русский текст хотя бы абзац текста на другом языке, то автоматические переносы в словах в нем будут делаться неправильно (или вообще окажутся невозможны), а полиграфические традиции языка, на котором написана вставка, будут нарушены. Как грамотно писать ЛАТЭХ'овские файлы на языках, отличных от русского и английского (в частности, как оформлять иноязычные вставки), рассказывается в приложении И.

Команды `\с`, `\'` и т.д. имеют один обязательный аргумент — букву, над (или под) которой надо ставить диакритический знак. Читатель может заметить противоречие со сказанным на с. 16: ведь обязательный аргумент должен быть заключен в фигурные скобки, а в нашей таблице в записях вроде `\'е` или `\"е` фигурных скобок нет. На самом деле противоречия нет, просто на с. 16 была сказана не вся правда<sup>2</sup>: если у ЛАТЭХ'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. 55). Так что можно, в принципе, не ставить букву в фигурные скобки и при командах наподобие `\с` 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.`

---

<sup>2</sup>Всей правды мы и сейчас не скажем.



## 5. Смена шрифтов в тексте

### 5.1. Предупреждение

Не увлекайтесь переключением шрифтов! Чем *меньше* различных видов шрифта использовано в тексте, тем легче его читать и тем красивее он выглядит.

### 5.2. Изменение начертания

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 ЛАТЭХ 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 ЛАТЭХ 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.), ЛАТЭХ выбирает шрифт за вас, и потому рекомендуется для выделения текста использовать в первую очередь именно эту команду.

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 ТЭХ'овских исходных текстов. Чтобы получить **шрифт без засечек**, надо воспользоваться командой `\textsf`. ШРИФТ «КАПИТЕЛЬ», В КОТОРОМ СТРОЧНЫЕ БУКВЫ ПРЕДСТАВЛЯЮТ СОБОЙ УМЕНЬШЕННЫЕ ПРОПИСНЫЕ, можно

Таблица III.1. Смена размера.

<code>\tiny</code>	Малюсенький $z$
<code>\scriptsize</code>	Очень маленький $z$ (как индексы)
<code>\footnotesize</code>	Маленький $z$ (как сноски)
<code>\small</code>	Мелкий $z$
<code>\normalsize</code>	Нормальный $z$
<code>\large</code>	Большой $z$
<code>\Large</code>	Очень большой $z$
<code>\LARGE</code>	Совсем большой $z$
<code>\huge</code>	Громадный $z$
<code>\Huge</code>	Гигантский $z$

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 прямым шрифтом знаки препинания (скобки и др.), встречающиеся среди текста, набранного курсивом:

<i>Рядом с <math>f(x)</math> (значением функции <math>f</math> в точке <math>x</math>) лучше использовать прямые скобки (а не курсивные).</i>	<code>\textit{Рядом с <math>f(x)</math></code> <code>\textup{(}значение функции</code> <code><math>f</math><code> в точке <math>x</math>\textup{)}}}</code> лучше использовать прямые скобки (а не курсивные)).</code>
---	--

### 5.3. Изменение размера

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 <code>\scriptsize</code> 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.2). В стандартных классах с основным шрифтом кегля 12 команды `\huge` и `\Huge` задают один и тот же размер (кегель 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. 125), 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.

Объясним мелким шрифтом, 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.

#### 5.4. Для любознательных: подробности о шрифтах

Каждый из доступных в  $\LaTeX$  текстовых шрифтов характеризуется следующими четырьмя атрибутами: семейством (*family*), насыщенностью (*series*),

начертанием (shape) и размером (size). Что такое размер, читатель разберется самостоятельно, а смысл остальных атрибутов следующий:

**семейство** означает «вариант начертания»; в стандартной поставке определены семейства `rmfamily` (шрифты с засечками), `sffamily` (шрифты без засечек) и `ttfamily` (шрифты типа «пишущая машинка»);

**насыщенность** определяет ширину и жирность шрифта. По умолчанию возможны насыщенности средняя (`mdseries`) и полужирная (`bfseries`);

**начертание** бывает: прямое (`upshape`), курсив (`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 ЛАТЭХ'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:

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.

Таблица III.2. Смена начертания: команды с аргументами и без.

Без аргументов	С аргументом	На печати выйдет
<code>{\rmfamily Шрифт}</code>	<code>\textrm{Шрифт}</code>	Шрифт
<code>{\sffamily Шрифт}</code>	<code>\textsf{Шрифт}</code>	Шрифт
<code>{\ttfamily Шрифт}</code>	<code>\texttt{Шрифт}</code>	Шрифт
<code>{\mdseries Шрифт}</code>	<code>\textmd{Шрифт}</code>	Шрифт
<code>{\bfseries Шрифт}</code>	<code>\textbf{Шрифт}</code>	<b>Шрифт</b>
<code>{\upshape Шрифт}</code>	<code>\textup{Шрифт}</code>	Шрифт
<code>{\itshape Шрифт}</code>	<code>\textit{Шрифт}</code>	<i>Шрифт</i>
<code>{\slshape Шрифт}</code>	<code>\textsl{Шрифт}</code>	<i>Шрифт</i>
<code>{\scshape Шрифт}</code>	<code>\textsc{Шрифт}</code>	ШРИФТ

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  $\text{\LaTeX}$  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 (`\textbf`) 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  $\text{\TeX}$ 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 VI.)

Таблица III.3. Смена начертания: старые команды.

Команда	Равносильна последовательности команд
<code>\bf</code>	<code>\normalfont\bfseries</code>
<code>\it</code>	<code>\normalfont\itshape</code>
<code>\sl</code>	<code>\normalfont\slshape</code>
<code>\sf</code>	<code>\normalfont\sffamily</code>
<code>\sc</code>	<code>\normalfont\scshape</code>
<code>\tt</code>	<code>\normalfont\ttfamily</code>
<code>\rm</code>	<code>\normalfont\rmfamily</code>

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 <i>and</i> I	You <code>{\itshape and}</code> I <code>\</code>
You <i>and</i> I	You <code>{\itshape and\/}</code> 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.

Все 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. 88), 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.

Наряду с описанными выше, в  $\text{\LaTeX}$ е пока сохраняются (для совместимости со старыми версиями) команды переключения шрифта, перечисленные 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).

## 5.5. 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:

```
cat                                \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. 101), чтобы `backslash` выглядел на печати как `\` (в большинстве `TeX`-овских шрифтов символу с этим кодом на печати соответствует нечто другое — можете попробовать и убедиться). Разумеется, `\symbol{'\s}` в этом примере — просто озорство, вместо этого можно было спокойно написать `s`.

## 6. Абзацы

Чтобы `TeX` сверстал абзац, никаких специальных усилий прикладывать 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 большая 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 (полиграфисты говорят «выключенными по формату»); 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 удастся, то получается аккуратный абзац, но нам сейчас интереснее, что происходит в том случае, когда это не удастся.

Прежде всего заметим, что «предел сжимаемости» строки не превышает  $\TeX$ 'ом ни при каких условиях<sup>3</sup>; что бы ни было, строки не станут более тесными, чем им позволяют параметры шрифта. Поэтому строки, которые не удалось ужать, остаются чересчур длинными (при этом, естественно, они выходят на поля). 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 записываются в `log`-файл — специальный файл с тем же именем, что у файла, который обрабатывался системой, и расширением `log` (при стандартном способе набора русских текстов, описываемом в этой книге, русские буквы запишутся в «виндовой» кодировке `cp1251`). Эти же сообщения выдаются на экран. При обработке русского текста русские буквы на экране могут оказаться нечитаемыми (в неправильной кодировке).

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, невзи-  
 рая 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. 19.) Далее сказано, что переполнение произошло при верстке абзаца (слова «`in paragraph`»), а затем указаны номера строк исходного файла, в которых был записан этот абзац.

<sup>3</sup>Если не предпринимать для этого специальных усилий.



Наконец, в этом сообщении приведен фрагмент неудачной строки вблизи ее конца (конца не в исходном тексте, а на печати)<sup>4</sup>. В некоторые слова вставлены дефисы: они показывают те места, в которых  $\TeX$  в принципе мог бы сделать переносы. Если взглянуть 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 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

Одна из важных причин *overfull*-ов состоит в том, что  $\TeX$ ’у попадает слово, которое он не знает, как перенести. Это может быть слово, каким-то образом ускользнувшее от его алгоритмов переноса, или слово, которое действительно перенести невозможно (например, «гвоздь»). Бывают и другие трудные случаи. Например, не будут, вообще говоря, автоматически переноситься слова, содержащие диакритические знаки (применительно к русским текстам — слова с проставленным ударением

---

<sup>4</sup>Загадочный набор символов, предшествующий фрагменту строки, характеризует текущий шрифт.

или с буквой ё, заданной как `\"е`), а также слова, 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). Наконец, бывает и так, что корень зла — не слово, а не поддающаяся переносу математическая формула. Что же делать в случаях, когда автоматически вставляемых `TeX`'ом переносов не хватает?

Совет номер один — попробуйте 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` 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`.

Наконец, если у вас подключен пакет `babel` с опцией `russian`, то в слове с дефисом можно вместо дефиса написать `"=` — тогда сохранится возможность переносить не только по дефису, но и там, где разрешается переносить его составные части (например, можно написать `генерал"=губернатор`).

Иногда в тексте встречаются пары слов, объединенные знаком `/`. По общим Т<sub>Е</sub>X'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 харак-  
терных черт систем типа  
UNIX/Linux.

Перенаправление ввода\slash вывода~---  
одна из характерных черт систем типа  
UNIX\slash Linux.

Заключительный совет: расставляйте сочетания `\-` не превентивно, а только в том случае, когда это действительно нужно для борьбы с переполнением. Т<sub>Е</sub>X'у все равно, а людям проще работать с исходным файлом, в котором меньше загадочных значков и слова легче читаются.

Бывают случаи, когда избавиться от переполнения не помогают даже идеально расставленные переносы. Что тогда делать, 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 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:

*разбиение абзаца на строки 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:

Черпаховый суп — изысканное деликатесное и диетическое блюдо	Черпаховый суп~--- изысканное деликатесное и диетическое блюдо <code>{\sloppy</code>
	<code>}</code>

А вот — типичная ошибка начинающего:

Черпаховый суп — изысканное деликатесное и диетическое блюдо	<code>{\sloppy</code> Черпаховый 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 значением 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 удастся,  $\TeX$  попыбует 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 <b>filename</b> задает имя файла.	Параметр <code>\mbox{\textbf{filename}}</code> задает имя файла.
--	--

Команда `\mbox` имеет один обязательный аргумент: в фигурных скобках может находиться любой текст, укладываемый 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 <code>\\</code> 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 <code>\linebreak</code> 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. 17): если в квадратных скобках указать какое-то расстояние (в  $\TeX$ -овских единицах длины — с. 19), 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:

<code>\smallskipamount</code>	маленький вертикальный пробел;
<code>\medskipamount</code>	вертикальный пробел побольше;
<code>\bigskipamount</code>	еще больше.

Точный размер этого пробела зависит от класса документа (и «классовых опций»); на с. 129 изображена величина соответствующих пробелов в стандартных классах со шрифтом кегля 11.

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 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 примере выше 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. 18; 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 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.	<code>badness = 0</code>
Как может выглядеть разреженная строка.	<code>badness = 142</code>
Как может выглядеть разреженная строка.	<code>badness = 740</code>
Как может выглядеть разреженная строка.	<code>badness = 1803</code>
Как может выглядеть разреженная строка.	<code>badness = 5147</code>
Как может выглядеть разреженная строка.	<code>badness = 10000</code>

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. 3.3, 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)<sup>5</sup>. 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  $r$ . При этом 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: все строки, для которых отношение  $r$  больше или равно 4.7 (примерно при этом значении получается 10000), рассматриваются `TeX`'ом как одинаково плохие.

<sup>5</sup>Кроме того, промежутки могут и сжиматься; пока речь идет только о жидких строках, это несущественно.



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 превосходит предела; на самом деле даже строки, мера разреженности которых не превосходит 200, выглядят все еще хорошо, хотя они уже и рассматриваются Т<sub>Э</sub>Х'ом как слегка разреженные: Т<sub>Э</sub>Х старается, чтобы такая «слегка разреженная» строка не попала в абзаце рядом со строкой, в которой промежутки между словами сжимались. Сообщения об underfull'e 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, — Т<sub>Э</sub>Х 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 Т<sub>Э</sub>Х выбирает между разреженной строкой и переполнением (см. с. 104)

При разбиении абзаца на строки Т<sub>Э</sub>Х *никогда* не создает строки, мера разреженности (badness) которых больше, чем значение Т<sub>Э</sub>Х'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.

Классический Т<sub>Э</sub>Х никогда не растягивал и не сжимал отдельное слово ради выравнивания строк; в его современных версиях, порождающих pdf-файлы, эти и другие «микротипграфические» эффекты также достижимы; см. документацию к пакету `microtype`.

В стандартном режиме значение параметра `\tolerance` равно 200. Если установить значение `\tolerance` равным 10000, т.е. максимально возможным, то может получиться так, что одна из строк абзаца окажется совершенно ужасной: Т<sub>Э</sub>Х вложит в нее «всю разреженность», чтобы не увеличивать то число, которое Т<sub>Э</sub>Х минимизирует при переборе различных вариантов разбиения абзаца на строки (грубо говоря, это число тем больше, чем больше 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 Т<sub>Э</sub>Х, с одной стороны, получит большую 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 Т<sub>Э</sub>Х 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 прежнее количество). 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. 108), 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 большее 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<sup>6</sup>, 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 можно, например, установить для абзацев, написанных на языке, отличном от основного языка документа, чтобы  $\TeX$  не сделал переносов во французском тексте по английским правилам. Впрочем, разумнее в этом случае воспользоваться средствами `babel`'я (см. приложение II).

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, 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. Борьбаться 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

---

<sup>6</sup>Не превышая значения `\tolerance`, разумеется!

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. 17); 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.

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}
```

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. 18).

Для длинных цитат, состоящих из нескольких абзацев, можно использовать окружение `quotation`. Оно полностью аналогично `quote`, за тем исключением, что в тексте, оформленном этим окружением, делаются абзацные отступы.

## 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. 110); 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. 108); 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. 18 приведен пример того, что может в этом случае получиться). Примеры:

левый  
марш

```
\begin{flushleft}
левый\\
марш
\end{flushleft}
```

	<code>\begin{flushright}</code>
наше дело	наше дело\\
правое	правое
	<code>\end{flushright}</code>
	 <code>\begin{center}</code>
а мы всегда	а мы всегда\\
в центре	в центре
	<code>\end{center}</code>

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 ЛАТЭХ'ом с обычным абзачным отступом, в противном случае — без отступа.

### 7.3. Стихи

В ЛАТЭХ'е существует окружение `verse`, предусмотренное для набора стихов. Строки в нем разделяются командой `\\`, а строфы — пустой строкой. Если строка окажется слишком длинной, она будет перенесена на следующую строку (не исключено, что в каком-то слове будет сделан перенос) и сдвинута примерно на 15 пунктов вправо. Никаких удобных средств для воссоздания встречающихся в реальных стихах графических эффектов в этом окружении не предусмотрено, и польза от него сомнительна: если набирать стихи с помощью окружений `flushleft`, `quote` или `center`, разделяя строки командой `\\`, а строфы, например, пустой строкой и командой `\smallskip` (см. с. 109 и 130 по поводу этих команд), получится не хуже.

Наличие или отсутствие абзачного 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  $\text{\LaTeX}$  выдаст сообщение об ошибке.

Вот как выглядел в исходном файле предшествующий текст:

```
\begin{itemize}
\item На печати каждый элемент перечня снабжается темным
кружочком (<<горох>> на жаргоне полиграфистов).
\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 перечня отмечаются полужирными
короткими тире, на третьем~--- звездочками, на четвертом~---
точками.
\item При попытке вложить пять таких окружений
\LaTeX{} выдаст сообщение об ошибке.
\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  $\text{\LaTeX}$  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.	<code>\begin{itemize}</code>
A Здесь мы сами задали заголовок.	<code>\item Этот элемент перечня помечен стандартно.</code>
999 Здесь тоже.	<code>\item[\sffamily A] Здесь мы сами задали заголовок.</code>
	<code>\item[999] Здесь тоже.</code>
	<code>\end{itemize}</code>

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  $\text{\LaTeX}$  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  $\text{\LaTeX}$ '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  $\text{\LaTeX}$  выдаст сообщение об ошибке.

В исходном тексте это выглядело так:

```
\begin{enumerate}
\item В окружении \texttt{enumerate} элементы списка нумеруются
цифрами или буквами.
\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
    выбираются автоматически.
  \end{enumerate}
\item На втором уровне элементы обозначаются
строчными буквами, на третьем --- римскими цифрами,
на четвертом --- прописными буквами.
\item При попытке вложить пять таких окружений
\LaTeX{} выдаст сообщение об ошибке.
\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. 20). 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. 121, 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. 17), представляющим собой этот заголовок. Пример.

<p><b>Поросенок:</b> хвост крючком, нос пяточком.</p> <p><b>Мышка:</b> под полом таится, кошки боится.</p>	<pre>\item[Поросенок:] хвост крючком, нос пяточком. \item[Мышка:] под полом таится, кошки боится.</pre>
--	---

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  $\text{\LaTeX}$  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).

**Перечни: итог.** 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. VI.2.5. Не сильно труднее сделать так, чтобы буквы, которыми нумеруются элементы нумерованного перечня, были русскими, а не латинскими, как в примерах в этой книге (в гл. VIII мы расскажем, как этого добиться). Можно менять оформление перечней и более серьезным образом, создавая перечни иного типа, чем рассмотренные выше. На данный момент наш  $\TeX$ нический уровень не столь высок, чтобы можно было освоить эти возможности  $\LaTeX$ 'а, но в гл. VIII будет рассказано и об этом.

## 7.5. Буквальное воспроизведение (`verbatim`, `verb`)

Окружение `verbatim` предназначено для буквального воспроизведения имеющихся в файле символов (шрифтом типа пишущей 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}`». При этом «закрывающую команду» `\end{verbatim}` следует писать в отдельной строке, ничего, кроме этого текста, не содержащей (для всех прочих  $\LaTeX$ -овских окружений это не обязательно). 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 `|` используется редко.

Команда `\verb|\dots|` задает многоточие. Знак `~\verb"|"` 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 L<sup>A</sup>T<sub>E</sub>X'овских и T<sub>E</sub>X'овских команд.

У команды `\verb` и окружения `verbatim` есть варианты «со звездочкой» (см. с. 18). 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 получите уйму сообщений об ошибке.

Если вы воспроизводите в режиме `verbatim` текст, простирающийся на многие страницы (например, компьютерную программу), то T<sub>E</sub>X'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 запись

```
\verbatiminput{something.txt}
```

равносильна

```
\begin{verbatim}
<содержимое файла something.txt>
\end{verbatim}
```

## 8. Сноски

Чтобы сделать сноску к какому-то месту в тексте, достаточно использовать команду `\footnote` с одним обязательным аргументом — текстом сноски. В стандартных классах L<sup>A</sup>T<sub>E</sub>X'а сноски<sup>7</sup> 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 ...
```

---

<sup>7</sup>Вроде этой.

В разделе VI.2, посвященном «счетчикам», мы расскажем о том, какие возможности есть для того, чтобы пометить сноски по-другому.

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<sup>2014</sup>. Предыдущий фрагмент выглядел в исходном тексте так:

```
обязательным\footnote[2014]{Вот какой ... }
```

При использовании команды `\footnote` с необязательным аргументом автоматическая нумерация сносок не сбивается: предыдущая сноска имела номер 7, затем мы искусственно создали сноску номер 2014, а следующая сноска<sup>8</sup> будет иметь номер 8.

В случае, если вы хотите сделать сноску к тексту, входящему в «блок» (например, в аргумент команды `\mbox`; в гл. VII 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:

```
Роман \mbox{‘‘Три\footnotemark\
мушкетера’’}\footnotetext{А не четыре!} написал Дюма.
```

В этом случае получится нормальная сноска, напоминающая нам, что в названии романа “Три<sup>9</sup> 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. 13).

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):

<sup>2014</sup> Вот какой интересный номер!

<sup>8</sup> Вот эта.

<sup>9</sup> А не четыре!

```
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 рассуждений: будем смотреть не только на строки и абзацы, но и на то, как они расположены на странице.

### 9.1. Понятие о режимах $\text{\TeX}$ 'а

Несколько упрощая ситуацию, можно сказать, что в процессе обработки исходного текста  $\text{\TeX}$  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)  $\text{\TeX}$  находится в горизонтальном режиме.
- Между абзацами, а также в начале работы (например, в процессе обработки преамбулы к  $\text{\LaTeX}$ 'овскому файлу)  $\text{\TeX}$  находится в вертикальном режиме.
- При обработке математических формул (см. гл. II)  $\text{\TeX}$  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. 13).

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  $\text{\TeX}$ 'овских команд, имеет смысл обозначить конец абзаца именно таким способом. Кроме того, команда `\par` бывает полезна в макроопределениях, о которых речь впереди.

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, ЛАТ<sub>Э</sub>X 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, Т<sub>Э</sub>X 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 Т<sub>Э</sub>X 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  $\TeX$ 'а (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  $\LaTeX$ 'а, 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.9), 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 предусматрива-



ют разное оформление страниц с четными и нечетными номерами — см. разд. IV.2 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  $\text{\LaTeX}$  немного обмануть: между двумя командами для разрыва страницы дать команду `\mbox{}`.

Наконец, существует команда `\pagebreak`, формально аналогичная команде `\linebreak` (см. с. 108). 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 обязательным 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 растёт 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  $\text{\TeX}$ -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 в принципе стоять не только кратное `\baselineskip`, но и любая длина, выраженная в Т<sub>Е</sub>X’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 Т<sub>Е</sub>X’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 Т<sub>Е</sub>X 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) Л<sup>A</sup>T<sub>Е</sub>X 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  $\smallskip$  промежуток;
- `\medskip` задает такой  $\medskip$  промежуток;
- `\bigskip` задает такой  $\bigskip$  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. 109.

Если вы хотите задать размер вертикального промежутка в явном виде, можно воспользоваться командой `\vspace`. Подобно команде `\hspace` (см. с. 92), 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 исчезающий 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 Т<sub>Е</sub>X'овских единицах, а `plus` и `minus` — так называемые «ключевые слова» Т<sub>Е</sub>X'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 Т<sub>Е</sub>X'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 Т<sub>Е</sub>X'ническим языком, команда `\vspace` вставляет в страницу «клей»<sup>10</sup>; расстояния, указанные после `plus`

<sup>10</sup>Свойства которого мало похожи на свойства настоящего клея. См. гл. VII.

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 ЛАТЭХ'ом, обладают растяжимостью и/или сжимаемостью, что помогает при нахождении оптимальных разрывов страниц.

Один частный случай растяжимых промежутков настолько важен, что в ЛАТЭХ'е для него предусмотрена специальная команда. Именно, в аргументе `\vspace` или `\vspace*` можно вместо длины, заданной в Т<sub>Е</sub>X'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`.

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`. В простых приложениях такие конструкции не встречаются. Мы еще будем говорить о них в разд. VII.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 все подробности в книге [2]). Можно, однако (и иногда это бывает необходимо), *пропорционально* увеличивать или уменьшать 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 необходима
\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. 107). Без команды `\selectfont` этот прием не сработает.

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. 130). 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. VI.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.2). 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 колонки 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  $\text{\TeX}$ -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  $\text{\TeX}$ , не сможет удовлетворить логически противоречивым требованиям; если ограничений на разрывы страниц слишком много, то  $\text{\TeX}$  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 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, стоит 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). Впрочем, книга, в которой выоты страниц «прыгают», смотрится весьма неважно, так что без крайней необходимости переключаться в такой режим не надо. Простой способ «поиграть» с высотой страниц — попробовать сделать какие-то внутритекстовые формулы выключными (или наоборот).




Режим, задающийся командой `\raggedbottom`, отключается с помощью команды `\flushbottom`.

## 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  $\text{\TeX}$ 'е линейкой (*rule* по-английски) называется любой черный прямоугольник. Для создания линеек в  $\text{\LaTeX}$ '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  $\text{\TeX}$ 'ом единицах измерения — см. с. 19). Линейка, созданная командой `\rule`, рассматривается  $\text{\TeX}$ '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. Т<sub>Е</sub>X'овские команды для генерации линеек

Рассмотренная нами Л<sub>А</sub>T<sub>Е</sub>X'овская команда `\rule` обладает рядом недостатков. Например, то обстоятельство, что создаваемые с ее помощью линейки воспринимаются Т<sub>Е</sub>X'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 абзачного отступа: Т<sub>Е</sub>X 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 знать, чему эта ширина равна, или, по крайней мере, как она обозначается). Избавиться от этого неудобства можно с помощью Т<sub>Е</sub>X'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 V, посвященной набору таблиц; в главе VII, посвященной «блокам», мы также встретимся с линейками.

## 11. Для любознательных: 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 `\hangindent=1cm \noindent`  
 отступ (по-английски Отрицательный абзацный отступ  
`hanging indentation`). (по-английски `hanging indentation`).

Здесь Т<sub>Е</sub>X'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 Т<sub>Е</sub>X'овский параметр, обозначаемый `\hangafter`:

Можно сделать так, чтобы от- `\hangindent=1cm \hangafter=2`  
 ступ начался не с первой стро- `\noindent` Можно сделать  
 ки, а там, где нам это так, чтобы отступ начался не с  
 потребовалось. 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  $1, 2, \dots, |n|$ . Пример:

С помощью рассмотрен- `\hangindent=1.5cm`  
 ных нами средств Т<sub>Е</sub>X'а `\hangafter=-3 \noindent`  
 можно выкапывать в аб- С помощью рассмотренных нами  
 зацах небольшие ямки (непонят- средств Т<sub>Е</sub>X'а можно  
 но, зачем). выкапывать в абзацах небольшие  
 ямки (непонятно, зачем).

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 вы- \hangindent=-2cm \hangafter=2  
копать ямку не слева, а справа. \noindent

Что ж, Т<sub>Е</sub>X позволя-  
ет сделать и так, бы-  
ло бы желание. Вскоре  
вы сможете убедиться,  
что и это — не предел.

На сей раз нам захотелось  
выкопать ямку 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 Л<sup>A</sup>T<sub>Е</sub>X'овских окружений наподобие itemize или quote: в таких окружениях Л<sup>A</sup>T<sub>Е</sub>X 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 Т<sub>Е</sub>X'а создать абзац совсем уж причудливой формы. Все переносы в словах и места разрывов строк были найдены Т<sub>Е</sub>X'ом автоматически. Получившийся абзац напоминает автору изображение шахматной ладьи. (Предложение в скобках нужно для того, чтобы ладья не выглядела, как обгрызенная мышами.)

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; 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  $\text{\TeX}$  считает, что эта формула занимает три строки, причем сама формула расположена в средней из этих трех (реально формула может, разумеется, занимать больше места).

## 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  $\text{\LaTeX}$  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) или даже свой собственный класс, весьма далекий от стандартных, но сейчас мы займемся стандартными классами.

### 1. Начнем с главного

Если вы пишете текст по-английски, то самой первой командой в файле должен быть `\documentclass` с обязательным и, возможно, необязательным аргументами (см. пример на с. 29; подробно мы это разберем в разд. 2). После этого можно подключать стилевые пакеты, менять стандартные параметры, определять или переопределять макросы... — и так до `\begin{document}`, после чего начинается собственно текст.

Если вы пишете по-русски, то после `\documentclass` надо *обязательно* написать

```
\usepackage[кодировка]{inputenc}
\usepackage[russian]{babel}
```

где вместо *кодировка* следует написать `cp1251`, `koi8-r`, `cp866` или `utf8`, в зависимости от того, какой из кириллических кодировок («виндовой», KOI8, «досовской», она же «альтернативная», или юникодом) задаются русские буквы (мы об этом уже несколько раз писали, но добрый совет не грех и повторить; повторим мы и рекомендацию узнать в какой-нибудь момент из разд. II.5 приложения II о других — и лучших —

корректных способах оформления русского текста). *Только после этого* можно приступать к загрузке стилевых пакетов, отличных от `babel`'я, и прочим приятным вещам. Русский текст, преамбула которого подключает пакет `babel` (указанным выше образом или так, как объясняется в разд. II.5), мы будем называть «должным образом оформленным» русским текстом.

Если вы пишете текст на языке, отличном от русского и английского, или многоязычный текст (два — это тоже много), то обратитесь к разд. II.4 приложения II.

## 2. Классы, пакеты и классовые опции

Команда `\documentclass`, с которой начинается любой L<sup>A</sup>T<sub>E</sub>X'овский файл, имеет один обязательный аргумент — название основного класса — и один необязательный, размещающийся перед обязательным, — список, через запятую, «классовых опций» (см. с. 13).

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[intlimits]{amsmath}
```

то у вас будут все эти возможности, плюс пределы интегрирования в выключных формулах будут располагаться над и под знаком интеграла.

Некоторые стилевые пакеты в своей работе, кроме того, учитывают «классовые опции» (это то, что указывается в необязательном аргументе команды `\documentclass` — см. ниже).

Каждый стиливой пакет можно подключать (в пределах одного документа) только один раз.

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.

Теперь вернемся к классам документов. В первой главе мы уже перечислили четыре основных класса, предоставляемых L<sup>A</sup>T<sub>E</sub>X'ом: **article**, **report**, **book**, и **proc**. Рассмотрим их подробнее. (Имеется еще важный класс **beamer**, предназначенный для создания презентаций — см. приложение П; Американское математическое общество распространяет также классы документов **amsart**, **amsproc** и **amsbook** — о них речь пойдет в приложении К.)

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 «±» означает «по умолчанию отсутствует, но будет присутствовать, если задать классовую опцию или специальную команду», знак «±» 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. 13), 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  $\text{\TeX}$  рассчитает размеры текста и полей так, чтобы они максимально соответствовали этому формату. Эти опции таковы (единица измерения указана в скобках):

`a4paper`  $210 \times 297$  (миллиметры) — самый ходовой в нашей стране;

`a5paper`  $148 \times 210$  (миллиметры);

`b5paper`  $176 \times 250$  (миллиметры);

`legalpaper`  $8.5 \times 14$  (дюймы);

`executivepaper`  $7.25 \times 10.5$  (дюймы).

Если ни одна из этих опций не указана,  $\text{\LaTeX}$  обычно считает, что размер бумаги равен (в дюймах)  $8.5 \times 11$  (этот формат бумаги иногда называют «letter»). Такой размер бумаги можно задать и явно, указав классовую опцию `letterpaper`.

Если предполагается расположить текст так, чтобы он шел параллельно широкому, а не узкому краю бумаги, то можно указать опцию `landscape`: в этом случае  $\text{\TeX}$  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  $90^\circ$  не повернет: он будет сверстан  $\text{\TeX}$ '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. 112), 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. 103) 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 классовых опции не указана «двусторонняя» печать (классовая опция `twoside`), а также классом `proc`. Во всех остальных случаях  $\text{\LaTeX}$  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  $\text{\LaTeX}$  не делает. По умолчанию в классе `report`  $\text{\LaTeX}$  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. 11).

По умолчанию при пользовании классом `proc` внизу каждой страницы будет напечатано слово `Page` («страница») и номер страницы. Если, в соответствии с рецептом на с. 141, подключить `babel` с опцией `russian`, то вместо «Page» будет печататься «с.», как в русских текстах и принято.

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`, а также пары команд `\[` и `\]` (для полноты добавим: а также с помощью не используемого на практике окружения `displaymath`, которое полностью эквивалентно `\[...\]`), 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.

### 3. Стиль оформления страницы

Для задания стиля оформления страницы в ЛАТЭХ'е предусмотрена команда `\pagestyle`. Эта команда имеет один обязательный аргумент — слово, обозначающее этот стиль. При использовании стандартными классами документов это слово должно быть одним из следующих:

<code>empty</code>	нет ни колонтитулов, ни номеров страниц;
<code>plain</code>	номера страниц ставятся внизу в середине строки, колонтитулов нет;
<code>headings</code>	присутствуют колонтитулы (включающие в себя и номера страниц);
<code>myheadings</code>	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 порождается ЛАТЭХ'ом автоматически, а задается пользователем в явном виде.

Если основным стиль — `article`, то по умолчанию страницы оформляются стилем `plain`, в двух других основных стилях — стилем `headings`. «Стиль» `myheadings` мы рассмотрим в разд. VIII.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:

<code>arabic</code>	арабские цифры (1, 2, 3,...)
<code>roman</code>	римские цифры (i, ii, iii,...)
<code>Roman</code>	римские цифры (I, II, III,...)
<code>alph</code>	строчные буквы (a, b, c,...)
<code>Alph</code>	прописные буквы (A, B, C,...)

Если вы пишете должным образом (т.е. так, как сказано в начале этой главы) оформленный русский текст, то есть еще две возможности:

**asbuk** строчные русские буквы (а, б, в, ...)  
**Asbuk** прописные русские буквы (А, Б, В, ...)

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. VIII.

## 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 Т<sub>Е</sub>X'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. 143 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 сверху страницы дополнительно будет пустое пространство, размер которого равен месту, отводимому на колонтитул (параметр `\headheight`, как мы узнаем из гл. VIII) 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. 143 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<sup>1</sup> 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 устанавливать таким образом, чтобы отношение

$$\frac{\textheight - \topskip}{\baselineskip}$$

было целым числом. В  $\text{\LaTeX}$ '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.

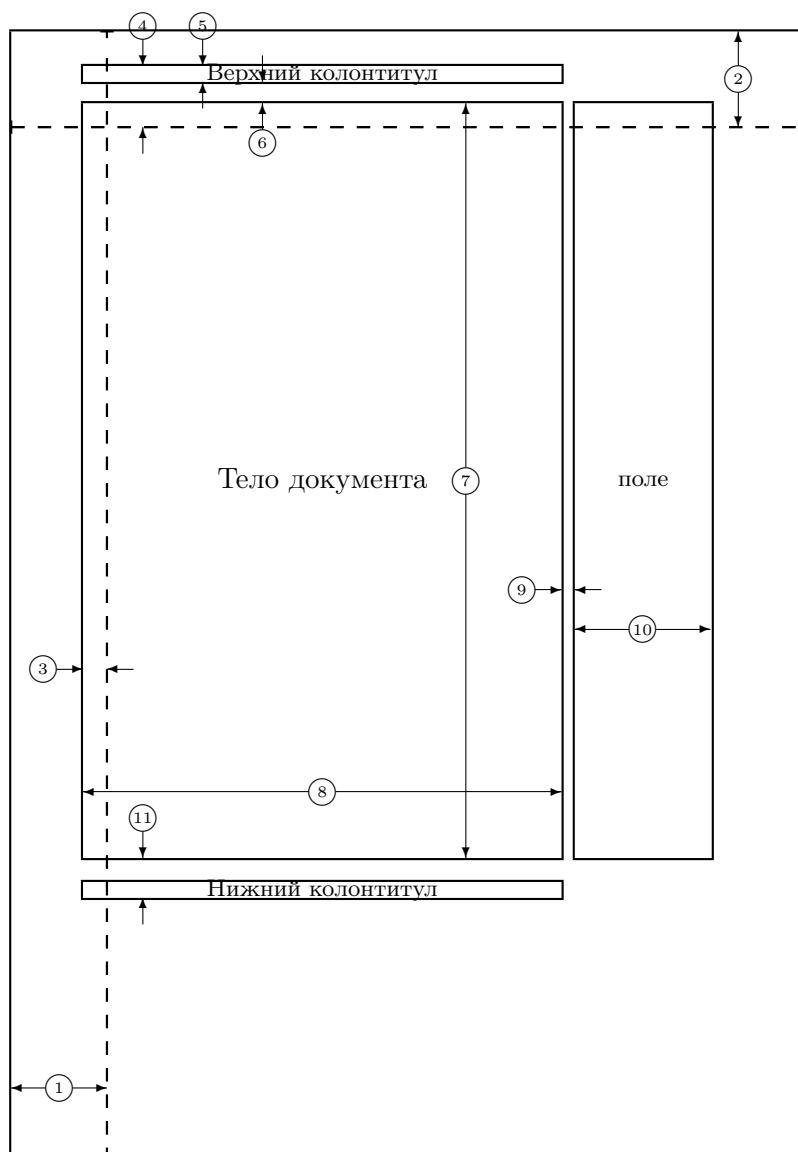
Можно и не знать точно этих размеров, но вычислить нужную величину `\textheight` средствами самого  $\text{\TeX}$ 'а: именно, если вы, скажем, хотите, чтобы на странице помещалось 40 строк, то напишите в преамбуле (после команды, изменяющей интерлиньяж, если вы таковой пользовались) следующее:

```
\setlength{\textheight}{40\baselineskip}
\setlength{\textheight}{\baselinestretch\textheight}
\addtolength{\textheight}{\topskip}
```

(см. разд. VI.4 по поводу `\setlength` и `\addtolength`).

Если вы запутались, посмотрите на изготовленную с помощью пакета `layout` картинку на с. 150, на которой изображена геометрия страницы  $\text{\LaTeX}$ '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.

<sup>1</sup>Точнее говоря, от ее базисной линии: см. гл. VII.



- |                          |                                    |
|--------------------------|------------------------------------|
| 1 1 дюйм + \hoffset      | 2 1 дюйм + \voffset                |
| 3 \oddsidemargin = -18pt | 4 \topmargin = -46pt               |
| 5 \headheight = 12pt     | 6 \headsep = 16pt                  |
| 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               |

Удобный интерфейс для установки параметров страницы предоставляется пакетом `geometry`; см. начало гл. VIII по поводу того, как раздобыть документацию к этому пакету (и сам пакет, если его нет в вашей поставке).

### 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`: у принтера, которым вы пользуетесь, могут быть свои представления о том, где находится край листа бумаги. Чтобы привести эти представления в соответствие с реальностью, можно изменить расположение страницы на бумаге. Для этого следует установить (в преамбуле) значения двух Т<sub>Е</sub>X'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.

## 5. Рубрикация документа

Работая с Л<sup>A</sup>T<sub>Е</sub>X'ом, разумно делать заголовки и нумерацию разделов документа не вручную, а с помощью специальных команд. 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. с. 160). Промежутки между разделами, их нумерация, те же колонтитулы — все это делается автоматически.

Кроме обязательного аргумента, у команды `\section` предусмотрен и необязательный, идущий перед обязательным; в нем записывается вариант заголовка, предназначенный для оглавления и колонтитулов (если класс предусматривает, что заголовок войдет в колонтитул).



```
\section[0 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. 20). 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  $\text{\LaTeX}$ ’а будет выдано сообщение о том, что метка неизвестна, а в дальнейшем, если номер помеченного раздела изменится,  $\text{\LaTeX}$  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. VIII.6 (см. с. 307).

У команды `\section` есть вариант «со звездочкой» (см. с. 18). 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  $\text{\LaTeX}$ ’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 во всех 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  $\text{\LaTeX}$  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. VIII, посвященной модификации стандартных классов.

### 5.3. Стандартные заголовки

Возможно, вы уже обратили внимание, что главы, создаваемые  $\text{\LaTeX}$ ’ом с помощью команды `\chapter`, называются «Chapter», если текст английский, и «Глава», если текст русский (должным образом оформленный). Это один из нескольких примеров «стандартных заголовков», которые  $\text{\LaTeX}$  генерирует автоматически. В нижеприведенной таблице указаны основные стандартные заголовки для английских и (должным образом оформленных) русских текстов. Вы вправе заменить эти заголовки на что-то еще по своему выбору. Если, например, вам хочется, чтобы заголовок рубрики `\chapter` выглядел по-русски не как «Глава», а как «Лекция», надо переопределить в преамбуле команду `\chaptername` следующим образом:

```
\renewcommand{\chaptername}{Лекция}
```

(по поводу `\renewcommand` и переопределения макросов см. гл. VI). В нескольких ближайших разделах мы объясним, как именно получить на печати упоминаемые в таблице список таблиц, список иллюстраций и т. п.

<code>\chaptername</code>	Chapter	Глава
<code>\contentsname</code>	Contents	Оглавление (или Содержание)
<code>\listfigurename</code>	List of Figures	Список иллюстраций
<code>\listtablename</code>	List of Tables	Список таблиц
<code>\abstractname</code>	Abstract	Аннотация
<code>\refname</code>	References	Список литературы
<code>\bibname</code>	Bibliography	Литература
<code>\indexname</code>	Index	Предметный указатель
<code>\figurename</code>	Figure	Рис.
<code>\tablename</code>	Table	Таблица
<code>\partname</code>	Part	Часть
<code>\appendixname</code>	Appendix	Приложение

Особых пояснений эта таблица, видимо, не требует. Если в классе определена команда `\chapter` (есть если это **book** или **report**), то в должным образом оформленных русских текстах оглавление так и называется «Оглавление», а если нет (в классах **article** и **proc**), то оно называется более скромно «Содержание», но для его переименования в обоих случаях надо переопределять команду `\contentsname`. С другой стороны, если класс документа есть **article** или **proc**, то для получения нестандартного названия списка литературы надо переименовать команду `\refname`, а если **report** или **book** — то команду `\bibname`. Пожалуйста, не перепутайте эти два случая, иначе L<sup>A</sup>T<sub>E</sub>X зафиксирует ошибку и выполнять вашу команду откажется. (Проще всего вообще ничего не переопределять без особой необходимости.)

#### 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» в английском тексте и «Аннотация» в должным образом оформленном русском тексте, если только вы 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 только следующее:

- 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).

## 5.5. Перемещаемые аргументы и хрупкие команды

Если в аргументе команды `\section` (или любой другой ЛАТЭХ'овской команды для рубрикации) присутствует не только текст, но и ТЭХ'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 ЛАТЭХ'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{0 свойства подмножеств пустых\protect\nolinebreak\
множеств}
...
\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. 109.) Если убрать в этом файле `\protect`, то на экране появится загадочное сообщение об ошибке.

Такого рода ситуация может возникать, когда ТЭХ'овская команда является частью текста, который будет записан в специальный файл и использован при следующем запуске ЛАТЭХ'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 ЛАТЭХ'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, но в большинстве случаев это и не понадобится: создатели современных версий ЛАТЭХ'а о вас

уже позаботились. Кстати, если в приведенном выше примере запретить разрыв не с помощью `\nolinebreak`, а с помощью обычного знака `~`, то никакого `\protect` и не понадобится.

## 6. Эпиграфы

Он знал довольно по-латыне,  
Чтоб эпиграфы разбирать.

*А. С. Пушкин*

Содержательно материал этого раздела относится скорее к предыдущей главе, но по  $\TeX$ ническим причинам обсуждать его приходится в связи с командами для рубрикации текста.

Грамотно оформить эпиграф к тексту с помощью элементарных средств  $\LaTeX$ 'а затруднительно. Лучше воспользоваться входящим в современные поставки стилевым пакетом `epigraph`.

Эпиграф к этому разделу мы получили следующим образом. Во-первых, в мы подключили стилевой пакет `epigraph` — иными словами, записали в преамбулу команду `\usepackage{epigraph}`. Во-вторых, перед началом раздела мы написали следующее:

```
\epigraph{%
Он знал довольно по-латыне,\\
Чтоб эпиграфы разбирать.}{%
\emph{А.\,С.\,Пушкин}}
```

Здесь знаки процента, разумеется, стоят для того, чтобы разрывы строк, сделанные ради читаемости кода, не воспринимались  $\TeX$ 'ом как пробелы. Как видите, у команды `\epigraph` два аргумента: первый — текст эпиграфа, второй — подпись под эпиграфом.

Если вы воспроизведете предложенный нами код у себя, то получится не совсем то, что вы видите в книге: шрифт окажется крупнее, а между текстом и подписью вы увидите горизонтальную черту («линейку» — см. разд. III.10). Именно так выглядит оформление эпиграфа по умолчанию. Русским полиграфическим традициям это не соответствует, так что мы в преамбуле, помимо подключения пакета `epigraph`, написали

```
\epigraphrule=0pt
```

(установили нулевое значение для параметра `\epigraphrule` со значением длины). Этот параметр — ширина «линейки», отличающей текст от подписи, если эта ширина становится равной нулю, то линейка, как видим, пропадает.

Далее, по умолчанию эпиграф печатается размером `small`, что, видимо, многовато. Чтобы установить нужный нам размер текста (мы хотели, чтобы это был `footnotesize`), пришлось переопределить команду `\epigraphsize` следующим образом:

```
\renewcommand{\epigraphsize}{\footnotesize}
```

Разумеется, и присваивание значений параметру `\epigraphrule`, и переопределение команды `\epigraphsize` можно делать в преамбуле только *после* подключения пакета `epigraph`: пока этот пакет не подключен, данные параметр и команда `TeX`’у неизвестны, так что при попытке их изменить вы получите сообщение об ошибке.

Иногда к одному разделу текста ставятся несколько эпиграфов. Для этих целей в пакете `epigraph` предусмотрено окружение `epigraphs` и команда `\qitem`, работающие следующим образом. Все эпиграфы собираются внутри окружения `epigraphs`, и каждый из них задается командой `\qitem`, устроенной точно так же, как `\epigraph`: в первом аргументе текст эпиграфа, во втором подпись. Например, эпиграфы ко второй главе того же «Евгения Онегина» можно было бы задать так:

```
\begin{epigraphs}
\qitem{0 rus!}{\emph{Hor.}}
\qitem{0 Русь!}{}
\end{epigraphs}
```

Поскольку у второго из этих эпиграфов, как известно, автор не указывается, второй аргумент у второй команды `\qitem` мы оставили пустым.

У команды `\epigraph` и окружения `epigraphs` имеется забавный побочный эффект, который необходимо учитывать: если их поставить в начале раздела документа (а где ж еще их ставить?), то следующий абзац будет начинаться с абзацным отступом, в то время как в `LaTeX`’е по умолчанию, как мы знаем, в первом абзаце раздела отступ подавляется. Чтобы не нарушать такой стиль оформления, после эпиграфа или эпиграфов, созданных с помощью пакета `epigraph`, абзацный отступ приходится подавлять вручную, начиная первый абзац собственно текста командой `\noindent`. Как говорят в таких случаях программисты, “It’s not a bug, it’s a feature”.

Место по горизонтали, отводимое под эпиграф, задается параметром со значением длины `\epigraphwidth`. Вы вправе прямо в преамбуле присвоить ему значение, которое считаете нужным (делать это надо опять-таки после подключения пакета `epigraph`: без этого `TeX` такого параметра, как `\epigraphwidth`, и знать не знает).

По умолчанию текст эпиграфа печатается выровненным влево. Если вам зачем-то нужно сделать его выровненным вправо или центрированным, переопределите команду `\textflush` на `flushright` или `center`:

```
\renewcommand{\textflush}{center}
```

Текст подписи к эпиграфу печатается выровненным вправо. Чтобы изменить это положение дел, можно аналогично переопределить на `flushleft` или `center` команду `\sourceflush`.

Вертикальные отступы перед эпиграфом и после него можно менять, изменяя значения параметров `\beforeepigraphskip` и `\afterepigraphskip`. Эти параметры представляют собой клей в Т<sub>Е</sub>Хническом смысле (см. гл. VII); их значения можно менять с помощью команды `\setlength`.

## 7. Титульный лист, оглавление, список литературы, аннотация

### 7.1. 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 Л<sup>A</sup>T<sub>E</sub>X'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 подходящие 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 Л<sup>A</sup>T<sub>E</sub>X'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. 116, а также пример на с. 18).

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  $\text{\LaTeX}$  напечатает на титуле дату своего запуска (как водится, по-английски, если текст английский, и по-русски, если текст русский и надлежащим образом оформлен).

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  $\text{\TeX}$ ’овской командой `\endgraf`).

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 А.\,А.\,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  $\text{\LaTeX}$ ’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  $\text{\LaTeX}$  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.

## 7.2. Оглавление

В процессе работы  $\text{\LaTeX}$  автоматически собирает информацию для создания оглавления и записывает ее в специальный файл с тем же именем, что у обрабатываемого файла, и расширением `toc`. Чтобы  $\text{\LaTeX}$  записал эту информацию, а затем воспользовался ею и напечатал оглавление, надо дать команду `\tableofcontents`.

Стало быть, оглавление, генерируемое  $\text{\LaTeX}$ ’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  $\text{\LaTeX}$  еще раз (напоминания об этом  $\text{\LaTeX}$  не выдаст).

Все оглавление в целом будет озаглавлено словом, определяемым командой `\contentsname` (см. разд. 5.3). Если русский текст оформлен должным образом (с использованием `\usepackage[russian]{babel}`), то корректное русское название оглавления напечатается «само собой».

Если вас не устраивает стандартный стиль оформления оглавления, прочтите в разд. VIII.4, как его можно изменить.

## 7.3. 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`<sup>2</sup>. 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 («библиографическая

---

<sup>2</sup>Если вам интересно, почему это окружение называется именно так, а не попросту `bibliography`, загляните в приложение Б; в нем же рассказано, как в некоторых случаях можно (полу)автоматизировать составление списка литературы.

метка»). В качестве такой метки 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.
- [4] А. П. Чехов. Медведь.

```

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.
\bibitem{Ch}
А.\,П.\,Чехов. Медведь.
\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 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,  $\text{\LaTeX}$  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\'etrie Alg\'ebrique.
...
\end{thebibliography}
```

После этого команда `\cite{Groth}` будет генерировать текст [EGA].

Списку литературы в целом  $\text{\LaTeX}$  автоматически дает заглавие, определяемое командой `\refname` в классах `article` и `proc` и `\bibname` в классах `report` и `book` (см. разд. 5.3).

## 8. Предметный указатель

В отличие от списка литературы, который при использовании описанных выше команд `\cite` и `\bibitem` получается совершенно автоматически, процесс создания указателя автоматизирован в  $\text{\LaTeX}$ е не полностью.

Именно, для создания предметного указателя надо сначала специальным образом пометить в файле термины, на которые вы собираетесь в предметном указателе ссылаться. При этом средствами  $\text{\LaTeX}$ 'а создается полуфабрикат (так называемый `idx`-файл), из которого предметный указатель получится после обработки отдельной программой, называемой обычно `makeindex`, входящей в настоящее время во все поставки  $\text{\LaTeX}$ 'а.

Мы расскажем именно об этом способе создания индекса, а в конце объясним, что можно сделать, если программа `makeindex`, описываемая ниже, вам по какой-то причине недоступна.

### Предупреждение

Те, кто собираются составлять индексы исключительно к английским текстам, могут этот раздел пропустить, но при работе с русскими текстами имейте в виду следующее. Если ваш  $\text{\LaTeX}$ 'овский файл написан по-русски с подключением пакета `inputenc` в преамбуле (как описано в начале этой главы), то `idx`-файл окажется нечитаемым и, что хуже, непригодным к обработке программой `makeindex` без дополнительных ухищрений. Поэтому если вы хотите пользоваться командами `\index` в русском тексте, оформляйте свой  $\text{\LaTeX}$ 'овский файл так, как описано в разд. И.5 приложения И. Тогда полуфабрикат индекса (`idx`-файл) окажется читаемым (и записанным в той же кодировке, что и ваш `tex`-файл). Если кодировка вашего исходного файла `cp1251` или `cp866` («альтернативная»), то после этого программа `makeindex` создаст

из `idx`-файла нормально отсортированный<sup>3</sup> индекс. Если вы работаете в кодировке `koï8-r`, то `idx`-файл будет в той же кодировке, и напрямую воспользоваться программой `makeindex` будет нельзя (как известно, в кодировке `koï8-r` порядок русских букв не совпадает с алфавитным). Впрочем, если вы пользуетесь кодировкой `koï8-r`, то почти наверняка у вас UNIX-подобная система, в которой ничего не стоит написать однострочный скрипт, данную проблему решающий (скажем, перед подачей на вход программы `makeindex` пропустить `idx`-файл через фильтр, преобразующий `koï` в `sr866`, а выходной поток `makeindex`'а пропустить через фильтр, осуществляющий обратное преобразование).

### 8.1. 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:

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.

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}
```

---

<sup>3</sup>За исключением буквы ё; на с. 168 написано, как добиться, чтоб ё сортировалась, как е.

Полученный таким образом `idx`-файл — это и есть полуфабрикат указателя, созданный `ЛATEX`-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 `ЛATEX`-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}
```

Теперь вернемся к нашему рассказу. 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. Наконец, команда `\indexspace` создает дополнительный вертикальный пробел (его можно использовать для отделения различных разделов указателя друг от друга):

компьютеры, 25–42	<code>\begin{theindex}</code>
IBM-совместимые, 28	<code>\item компьютеры, 25--42</code>
ремонт, 35	<code>\subitem IBM-совместимые, 28</code>
цены, 30	<code>\subsubitem ремонт, 35</code>
болгарские, 26	<code>\subsubitem цены, 30</code>
принтеры, 40	<code>\subitem болгарские, 26</code>
	<code>\item принтеры, 40</code>
кошки, 120	<code>\indexspace</code>
собаки, 140–156	<code>\item кошки, 120</code>
	<code>\item собаки, 140--156</code>
	<code>\end{theindex}</code>

Предметный указатель, получаемый из окружения `theindex`, печатается `ЛATEX`-ом в две колонки (даже тогда, когда сам документ печатается в одну колонку). Кроме того, `ЛATEX` автоматически дает указателю заглавие, определяемое командой `\indexname` (см. разд. 5.3).

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` (подрисовочная подпись; см. следующий раздел) и т. п.

## 8.2. Простейшие средства

В простейшем случае программа `makeindex` вызывается так:

```
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. 164, 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:

```
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 L<sup>A</sup>T<sub>E</sub>X'ом получится idx-файл<sup>4</sup>; 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
```

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 T<sub>E</sub>X'овской команды `\`", можно (и нужно) записывать без предосторожностей:

---

<sup>4</sup>В том, конечно, случае, если в преамбуле была команда `\makeindex`.

```
\index{кавычка ("")}  
\index{ёлочка}  
\index{"ежик}
```

При этом соответствующие записи в `ind`-файле могут получиться такими:

```
\item "ежик, 12  
\item ёлочка, 8  
\item кавычка ("), 6
```

### 8.3. 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. 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 то, что написано *левее* него, а в `ind`-файл записывается то, что *правее* него (мнемоническое правило для знающих английский: символ `@` рассматривать как сокращение от “actually”). 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}!игруана@\textbf{игруана}}
```

Этот же прием позволяет добиться, чтобы при сортировке слов с буквой ё эта буква приравнивалась к е: вместо `\index{берёзка}` надо писать `\index{березка@берёзка}`.

Напоминание: если перед `@` или `|` стоит кавычка `"`, то эти значки рассматриваются просто как символы.

Программа `makeindex` может оформлять номера разных страниц по-разному. 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}
```

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.5.4).

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}
```

## 8.4. 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 **s**», то есть сказать

```
makeindex -s 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 **s**, но это не обязательно):

```
makeindex -s -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.

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).

## 8.5. Если программы `makeindex` нет

На худой конец можно обойтись не только без программы `makeindex`, но и без команд `\index`: 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 текст, то можно попросту напечатать весь документ, кроме указателя, и вручную выписать требуемые номера страниц<sup>5</sup>.

<sup>5</sup>Можно также попробовать пометить все места, на которые надо сослаться, с помощью команды `\label`, а в окружении `theindex` получить номера страниц с помощью `\pageref` — лишь бы `TeX`'у хватило памяти.

Тем не менее сгенерировать `idx`-файл и работать далее с ним всяко удобнее. Если нет возможности обработать этот файл программно, можно, по крайней мере, сделать следующее.

Во-первых, надо средствами текстового редактора отсортировать строки `idx`-файла и слить строки с одинаковым термином. После этого 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}
```

Теперь  $\TeX$  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. VI, в которой подробно рассмотрен процесс определения новых команд, вы поймете, почему этот рецепт работает.

## 9. Иллюстрации и таблицы

### 9.1. Включение графики

Когда создавался  $\TeX$ , компьютерная графика только начинала развиваться. Поэтому никакого стандартизированного способа включить графику в текст в  $\TeX$ 'е предусмотрено не было. Однако же создатель  $\TeX$ 'а Дональд Кнут предусмотрел механизм, позволяющий записать в `dvi`-файл (получающийся, как мы помним, в результате обработки  $\TeX$ -файла) любую дополнительную информацию — в расчете на то, что в дальнейшем этот механизм можно будет использовать и для подключения графических файлов. Так оно в итоге и произошло: в настоящее время существует механизм для работы с графикой в  $\LaTeX$ 'е, являющийся стандартом *de facto*.

Для включения графики в текст используется пакет **graphicx** (буква **x** в названии пакета — не опечатка). Если он подключен, то для вставки рисунка надо написать так:

```
\includegraphics{имя_графического_файла}
```

В графических файлах зачастую содержится информация о размерах рисунка; если  $\text{\LaTeX}$ ’у удастся ее считать, то картинка, включаемая в текст с помощью `\includegraphics`, будет рассматриваться  $\text{\TeX}$ ’ом как одна большая «буква» указанных размеров. Поэтому если использовать `\includegraphics` с картинкой нормального размера, то картинка раздвинет соседние строки и результат будет выглядеть нелепо. Ясно, что естественное место иллюстрации к тексту — между абзацами (см. особенно разд. 9.3).

С картинкой, вставляемой в текст с помощью `\includegraphics`, можно проделывать различные манипуляции, задающиеся с помощью необязательного аргумента этой команды. В данном случае необязательный аргумент ставится *перед* обязательным. Перечислим, что именно можно сделать.

Во-первых, изображение можно пропорционально увеличить или уменьшить: если, скажем, написать

```
\includegraphics[scale=0.5]{имя_файла}
```

(вокруг знака равенства можно оставлять пробелы), то линейные размеры изображения уменьшатся вдвое. Другой способ пропорционально уменьшить или увеличить картинку — в явном виде задать ее ширину или высоту:

```
\includegraphics[width=4cm]{имя_файла}
```

или

```
\includegraphics[height=1in]{имя_файла}
```

В любом из этих случаев, если задать один из этих линейных размеров, то другой также изменится таким образом, чтобы не нарушить пропорции. Размер можно задавать в любых  $\text{\TeX}$ ’овских единицах длины.

Если задать в явном виде и длину и ширину, например, так:

```
\includegraphics[width=4cm,height=1in]{имя_файла}
```

(после запятой также можно оставить пробелы), то ширина и высота изображения будут такими, как указано, а само изображения при этом подвергнется соответствующему растяжению или сжатию. Если же, помимо высоты и ширины, написать еще, как в следующем примере, волшебное слово `keepaspectratio`

```
\includegraphics[width=4cm,height=1in,  
keepaspectratio]{имя_файла}
```

то пропорции изображения все же сохраняются; если отношение затребованной высоты к затребованной ширине не совпадает с отношением реальным, то из двух заказов на линейный размер будет выполнен тот, для которого, при сохранении пропорций, рисунок окажется меньше.

Рисунки можно поворачивать. Для этого в необязательном аргументе команды `\includegraphics` указывается угол поворота (в градусах, против часовой стрелки). Например, если написать

```
\includegraphics[height=1in, angle=45]{имя_файла}
```

то изображение сначала будет смасштабировано так, что его высота станет равной одному дюйму, а затем повернуто на 45°.

Наконец, из рисунков можно вырезать прямоугольники (сами графические файлы при этом не меняются, но напечатано будет только то, что находится внутри указанного вами прямоугольника). Для этого удобно использовать слова `trim` и `clip`. Если написать

```
\includegraphics[trim=1in 2in 0in 3in, clip]{имя_файла}
```

(не забудьте про `clip`, иначе ничего не получится), то на печати отображение будет обрезано на 1 дюйм с левого края, на 2 дюйма с нижнего края, вообще не будет обрезано с правого края и будет обрезано на 3 дюйма с верхнего края — необходимо указать все четыре параметра (через пробелы, а не запятые), именно в таком порядке: левый, нижний, правый, верхний. Можно использовать любые Т<sub>Е</sub>X'овские единицы длины. Перед `trim` в необязательный аргумент команды `\includegraphics` можно поместить указания по изменению масштаба (например, с помощью `scale`) и по повороту рисунка (с помощью `angle`).

Если у команды `\documentclass` или у самого пакета `graphicx` указана опция `draft`, то вместо рисунков будут печататься белые прямоугольники тех же размеров, а в них — имена соответствующих графических файлов.

До сих пор мы обходили молчанием вопрос о том, с какими форматами графических файлов умеет работать пакет `graphicx`. Ответ на этот вопрос зависит от того, какой программой мы собираемся обрабатывать наш Л<sup>A</sup>T<sub>Е</sub>X'овский файл.

Если вы используете для этих целей «классический» Л<sup>A</sup>T<sub>Е</sub>X, генерирующий из Т<sub>Е</sub>X-файлов файлы с расширением `.dvi` (соответствующая программа обычно называется `latex`), то лучше всего пользоваться файлами в формате EPS, он же Encapsulated PostScript (обычно такие файлы имеют расширение `.eps`) или файлами, порождаемыми программой MetaPost (см. приложение М) — их формат близок к EPS, и обычно они имеют расширение вида `.1`, `.2` и т. п. Можно также использовать некоторые растровые форматы (например, `bmp`).

Если же вы обрабатываете свои  $\text{T}_{\text{E}}\text{X}$ -файлы с помощью программы, генерирующей  $\text{pdf}$ -файлы (обычно она называется  $\text{pdf}_{\text{L}}\text{a}_{\text{T}}\text{E}_{\text{X}}$ ), то репертуар возможных графических файлов расширяется. В первую очередь можно использовать, естественно, графические файлы в формате  $\text{PDF}$ . Файлы формата  $\text{EPS}$  и файлы, генерируемые  $\text{MetaPost}$ 'ом, тоже подходят, но для последних необходимо, чтобы они имели расширение  $\text{.mps}$  (на худой конец файлы можно просто переименовать). Наконец, во многих случаях удастся обработать и большинство растровых форматов, включая  $\text{JPEG}$ .

Помимо уже упоминавшегося  $\text{MetaPost}$ , о котором мы расскажем в приложении М, в настоящее время для создания графических файлов, хорошо интегрирующихся с  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 'ом, используют также такие средства, как  $\text{tikz}$  и  $\text{asymptote}$  — в этой книге мы о них не рассказываем, но они также распространяются свободно и доступны в сети, вместе с обширной документацией.

Если вам позарез нужно сделать картинку, а воспользоваться средствами для создания нормального графического файла вы по той или иной причине не можете, то иногда (если рисунок *очень* простой) помогает такое архаичное средство, как  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 'овское окружение  $\text{picture}$ . См. приложение Р.

При размещении картинок между абзацами может случиться, что иллюстрация попадет на разрыв страниц, что неприемлемо. Ниже описывается  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 'овский механизм, позволяющий более или менее автоматизировать борьбу с такого рода неприятностями.

## 9.2. Плавающие объекты

Чтобы разместить в тексте иллюстрацию, удобно воспользоваться окружением  $\text{figure}$ . Стоящий между  $\backslash\text{begin}\{\text{figure}\}$  и  $\backslash\text{end}\{\text{figure}\}$  текст автоматически размещается  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ '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  $\text{figure}$  называют еще «плавающая иллюстрация». Подчеркнем, что между  $\backslash\text{begin}\{\text{figure}\}$  и  $\backslash\text{end}\{\text{figure}\}$  может быть не только команда  $\backslash\text{includegraphics}$ , но и совершенно любой текст и формулы, допустимый с точки зрения  $\text{T}_{\text{E}}\text{X}$ 'а. Все то, что находится внутри окружения  $\text{figure}$ , будет заведомо напечатано на одной странице.

Команда  $\backslash\text{caption}$  позволяет сделать подрисуючную подпись. Эта команда имеет один обязательный аргумент — текст подписи. На печати подпись состоит из слова, определенного командой  $\backslash\text{figurename}$  («Figure», если не подключать  $\text{babel}$ , и «Рис.» в должным образом оформленном русском тексте — см. разд. 5.3), порядкового номера



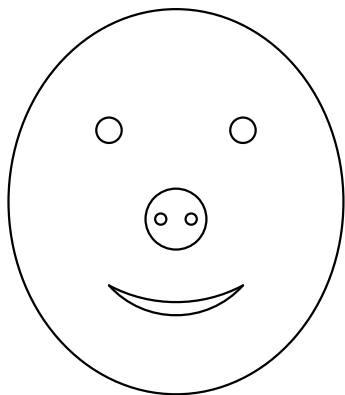


Рис. IV.1. Свинка без ушей

иллюстрации, присвоенного ей  $\text{\LaTeX}$ ’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. Естественно 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.1 на с. 176 получился так:

Например, `рис.~\ref{piggy}` на с.~\pageref{piggy} ...

```
\begin{figure}
\includegraphics[scale=2]{pig.mps}
\caption{Свинка без ушей}\label{piggy}
\end{figure}
```

Свинка смотрелась бы лучше, если бы была расположена по центру; на худой конец для этого можно использовать окружение `center`, но лучше — приемы, описываемые нами в гл VII.

Внутри одного окружения `figure` команд `\caption` может быть несколько, 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.1 после номера стоит точка. К сожалению, согласно  $\text{\LaTeX}$ ’овскому стандарту в этом месте ставится не точка, а двоеточие, что в русском тексте выглядит неудачно. В главе VIII объясняется, как с этим можно бороться.

У окружения **figure** предусмотрен необязательный аргумент, с помощью которого можно высказать L<sup>A</sup>T<sub>E</sub>X'у свои пожелания по поводу размещения иллюстрации в тексте. Именно, после `\begin{figure}` (без 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**, с двумя отличиями: подпись, генерируемая командой `\caption`, начинается со слова, определенного в команде `\tablename` (так что переопределять, при необходимости, надо именно эту команду — см. разд. 5.3), и 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. 160): материал для этих списков собирается в специальные файлы с расширениями `lof` (для иллюстраций) и `lot` (для таблиц); при каждом запуске  $\text{\LaTeX}$ 'а информация, записанная в этих таблицах, относится к предыдущему запуску, так что в самом конце может понадобиться запустить  $\text{\LaTeX}$  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  $\text{\LaTeX}$ 'е, мы расскажем в гл.V.

В разд. VIII.7 мы расскажем о том, как можно модифицировать оформление плавающих иллюстраций и таблиц.

### 9.3. Рисунки в оборку

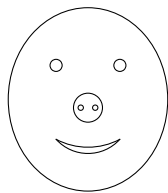


Рис. IV.2.

Окружения `figure` и `table` определяют иллюстрации и таблицы, простирающиеся на всю ширину текста, и ничего иного в стандартном комплекте  $\text{\LaTeX}$ 'а не предусмотрено. Пользователями  $\text{\LaTeX}$ 'а разработано несколько стилевых пакетов, позволяющих с бóльшим 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, хотя, конечно, размещение обтекаемых текстом иллюстраций полностью автоматизировано быть не может и всегда требует определенной ручной работы.

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 Т<sub>E</sub>X’овских единицах длины или выраженную через Т<sub>E</sub>X’овские параметры со значением длины). Например, код, задающий рис. IV.2, мог выглядеть примерно так:

```
\begin{wrapfigure}{o}{62.2pt}
  \langle команды, задающие рисунок \rangle
\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. 144, 147); 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, Л<sub>A</sub>T<sub>E</sub>X 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 Т<sub>E</sub>X делает в этом абзаце разрывы строк, 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 большим 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 L<sup>A</sup>T<sub>E</sub>X’a пакет `wrapfig` может и не входить; тогда его придется доставать из интернета. См. приложение О по поводу того, где в интернете хранятся T<sub>E</sub>X’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 понятно, как мы надеемся, все, кроме одного: откуда узнать ширину рисунка? Для ответа на этот вопрос нам придется опять (в который уже раз) забежать вперед и воспользоваться некоторыми конструкциями из гл. VI. Из этой главы (а именно, из параграфа VI.4) можно узнать, что L<sup>A</sup>T<sub>E</sub>X позволяет создавать свои собственные параметры со значением длины (см. разд. I.2.9) наподобие `\parindent`, а также что можно измерить ширину любого фрагмента текста с помощью команды `\settowidth`. Руководствуясь этим, поступим следующим образом. В преамбуле определим новый параметр со значением длины, специально предназначенный для результатов измерения ширины обтекаемых иллюстраций. Для него нужно придумать какое-то «незанятое» имя — пусть это будет `\WD`:

```
\newlength{\WD}
```

Теперь перед окружением `wrapfigure` мы измерим (с помощью команды `\settowidth`) ширину нашей картинке и запишем результат измерения в параметр `\WD`, а затем передадим `\WD` (а не явное значение размера) в качестве аргумента окружению `wrapfigure`. Вот как это можно сделать для нашего рисунка IV.2:

```
\settowidth{\WD}{\includegraphics{piggy.eps}}
\begin{wrapfigure}{o}{\WD}
\includegraphics{piggy.eps}
\caption{} \label{piggy}
\end{wrapfigure}
```

#### 9.4. Как влиять на расположение плавающих объектов

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 посередине иллюстрации должно быть место разрыва страницы), то

L<sup>A</sup>T<sub>E</sub>X действует так, словно в необязательном аргументе стояло не **h**, а **ht**. В результате иллюстрация будет напечатана вверх 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 на той странице, на которую эта команда попала. Можно применить команду и с необязательным аргументом: если написать

```
\suppressfloats[t]
```

то вверх данной страницы иллюстрации размещаться не будут; если в качестве необязательного аргумента указать **b**, то иллюстрации заведомо не появятся внизу данной страницы.

С другой стороны, L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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. VIII.7.

## 10. Еще о метках и ссылках

В разных местах этой книги уже шла речь о том, как можно пометить различные места документа, а затем на эти помеченные места сослаться. В настоящем разделе мы систематизируем эту информацию. Дополнительные сведения о том, как можно влиять на вид ссылок, создаваемых командой `\ref`, читатель найдет в разд. VI.2.

### 10.1. 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 ЛАТЭХ'a 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. VI.3; можно также самостоятельно создавать команды, дающие автоматическую нумерацию (см. гл. VI). 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 ЛАТЭХ'a 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, ЛАТЭХ 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 ЛАТЭХ'a (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 символы `~` («тильда») или `\` («backslash»). Прочие символы в аргументе команды `\label` (включая пробел или русские буквы) иногда безобидны, а иногда приводят к синтаксическим ошибкам. Пока вы не стали  $\TeX$ ником, лучше такие эксперименты не ставить.

## 10.2. Визуализация меток

Возможность автоматической генерации ссылок, предоставляемая командами `\label` и `\ref`, — большое благо, но всякое техническое усовершенствование приносит и новые проблемы. Предположим, что, рассматривая пробную распечатку, вы решили добавить ссылку на формулу 3.7. Писать прямо «3.7» не следует (в процессе дальнейшей работы над текстом номер формулы может измениться), но как вспомнить метку, которой вы неделю назад обозначали эту формулу?

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.<sup>6</sup> 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 белой распечаткой строку

```
\usepackage{showkeys}
```

из преамбулы документа надо удалить.

Надо сказать, что в современных реализациях  $\TeX$ 'а существует и более простой способ решить эту задачу: обычно программу для просмотра `dvi`-файлов и ваш любимый текстовый редактор можно совместно настроить таким образом, что если щелкнуть мышью по какому-то месту в просматриваемом `dvi`-файле, то в окне с редактором курсор окажется (примерно) в соответствующем месте исходного текста. При использовании такой технологии (она называется «reverse search» — обратный поиск) можно в принципе обойтись и без пакета `showkeys`.

---

<sup>6</sup>На верстку эти надпечатки не влияют.



### 10.3. Технические подробности мелким шрифтом

Скажем несколько слов о том, как происходит автоматическая генерация ссылок. Когда в обрабатываемом файле встречается команда `\label`,  $\text{\LaTeX}$  записывает информацию о ней в специальный файл, называемый `aux`-файлом (при обработке файла `text.tex` имя `aux`-файла будет `text.aux`)<sup>7</sup>. 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  $\text{\TeX}$ -ническим языком, это вид на печати того счетчика, который последним подвергся увеличению с помощью `\refstepcounter` — см. гл. VI).

Далее, всякий `aux`-файл читается  $\text{\LaTeX}$ -ом за один сеанс работы дважды: первый раз до начала обработки текста и второй раз — после ее завершения. При первом чтении `aux`-файла  $\text{\LaTeX}$  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)  $\text{\LaTeX}$  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».

## 11. 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`),

---

<sup>7</sup>Для каждого из файлов, включаемых в текст с помощью команды `\include`, создается отдельный `aux`-файл.

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 случае). Если документ печатается в две колонки, то заметка всегда выводится на 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. 44 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 (L<sup>A</sup>T<sub>E</sub>X 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

# Печать текста с выравниванием

Так как полиграфические шрифты являются, как правило, «пропорциональными» (каждая буква имеет свою ширину), добиться выравнивания колонок в таблицах, просто считая буквы, невозможно. В этой главе мы рассмотрим два основных способа, предоставляемых ЛАТ<sub>E</sub>X'ом для печати текста с выровненными колонками (например, таблиц): сначала более элементарный, но требующий больше ручного вмешательства, затем более автоматизированный. Впрочем, печать таблиц (в любой издательской системе) — дело всегда непростое, редко когда позволяющее обойтись без ручной доработки.

## 1. Окружение `tabbing`

### 1.1. Элементарные средства

Табулятор имитируется в ЛАТ<sub>E</sub>X'е с помощью окружения `tabbing`. При печати таблиц с помощью этого окружения пользователь сам задает места, в которых должна начаться очередная колонка. (Читателям старшего поколения это может напомнить так называемый табулятор на пишущих машинках, про которые тоже, видимо, пора писать «так называемые».) 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	<code>\begin{tabbing}</code>
раз	два	три	<code>начало\quad\=середина%</code>
раз	два	три	<code>\quad\=конец\\</code>
начинаем	продолжаем	заканчиваем	<code>раз\&gt;два\&gt;три\\</code>
			<code>раз\&gt; два\&gt; три\\</code>
			<code>начинаем\&gt;продолжаем\&gt;</code>
			<code>заканчиваем\\</code>
			<code>\end{tabbing}</code>

В первой строке мы задали две позиции табуляции двумя командами `\=`. 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). 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: 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.

Кроме установки дополнительных интервалов экспериментальным путем, можно расставлять позиции табуляции, пользуясь какой-нибудь строкой в качестве образца. Именно, если закончить строку не командой `\\`, а командой с суровым названием `\kill`, то напечатана эта строка не будет, но все позиции табуляции, установленные в ней, будут запомнены  $\text{\LaTeX}$ ’ом, и их можно будет использовать в последующих строках. В приведенном выше примере можно было бы написать так:

начало	середина	конец	<code>\begin{tabbing}</code>
раз	два	три	<code>начинаем \=продолжаем \=</code>
начинаем	продолжаем	заканчиваем	<code>заканчиваем\kill</code>
			<code>начало\&gt;середина\&gt;конец\\</code>
			<code>\bfseries раз\&gt;</code>
			<code>\itshape два\&gt;три\\</code>
			<code>начинаем\&gt;продолжаем\&gt;</code>
			<code>заканчиваем\\</code>
			<code>\end{tabbing}</code>

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. 94). Команды `\'` и `\'` также имеют внутри этого окружения особый смысл (см. ниже). Поэтому, если внутри `tabbing` нам понадобился диакритический знак (скажем, над буквой *e*), то надо набирать `\a=e` вместо `\=e` (и аналогично для `\'` и `\'`).

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.

## 1.2. Переустановка позиций табуляции.

Команды `\=`, устанавливающие позиции табуляции, можно давать не только в первой строке. Именно, 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 `\=`.

Пример:

парочка позиций табуляции	<code>\begin{tabbing}</code>
плюс еще одна здесь:	парочка <code>\=позиций</code>
теперь их уже три	<code>\=табуляции\\</code>
Вторую мы сменим	<code>\&gt;плюс\&gt;еще</code>
где эти позиции	одна здесь: <code>\=\\</code>
	теперь <code>\&gt;их\&gt;</code>
	уже <code>\&gt;три\\</code>
	Вторую <code>\&gt;мы\quad</code>
	<code>\=сменим \&gt;</code>
	и посмотрим: <code>\\</code>
	где <code>\&gt;эти\&gt;</code>
	позиции <code>\&gt;теперь\\</code>
	<code>\end{tabbing}</code>

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:

	<code>\begin{tabbing}</code>
раз два три четыре	<code>раз\quad\=два\quad\=три\quad\=четыре\\</code>
гиппопотам аллигатор	<code>\pushtabs гиппопотам\quad\=аллигатор\\</code>
раз два	<code>раз\&gt;два\\ три\&gt;четыре\\</code>
три четыре	<code>\poptabs</code>
one two three four	<code>one\&gt;two\&gt;three\&gt;four\\</code>
viens divi trīs četri	<code>viens\&gt;divi\&gt;tr\=a\=i s\&gt;\v{c}etri\\</code>
	<code>\end{tabbing}</code>

Команды `\pushtabs` и `\poptabs` должны быть «сбалансированы»: каждой команде `\pushtabs`, запоминающей позиции табуляции, должна

соответствовать вспоминая их команда `\poptabs`. Если это условие не выполнено, вы получите сообщение об ошибке. Обратите также внимание, что знак долготы над буквой *i* в слове `tīs` («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	<code>\begin{tabbing}</code>
<code>à gauche</code>	<code>à droite</code>	<code>\hspace{3.5cm}\=\kill</code>
<code>links</code>	<code>rechts</code>	<code>слева\&gt;справа\'\'</code>
<code>pa kreisi</code>	<code>pa labi</code>	<code>\a'a gauche\&gt;\a'a droite\'\'</code>
		<code>links\&gt;rechts\'\'</code>
		<code>pa kreisi\&gt;pa labi\'</code>
		<code>\end{tabbing}</code>

Еще раз обратите внимание, что для постановки диакритического знака над буквой *a* нам пришлось писать `\a'` вместо `\'` (см. с. 188).

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	<code>\begin{tabbing}</code>
<code>à gauche</code>	<code>à droite</code>	<code>слева\ 'справа\'</code>
<code>links</code>	<code>rechts</code>	<code>\a'a gauche\ '\a'a droite\'</code>
<code>pa kreisi</code>	<code>pa labi</code>	<code>links\ 'rechts\'</code>
		<code>pa kreisi\ 'pa labi\'</code>
		<code>\end{tabbing}</code>

Кстати, здесь нам вообще не понадобилось устанавливать позиции табуляции. Впрочем, соотрится 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	<code>\begin{tabbing}</code>
	два			<code>раз \=два \=три \=\kill</code>
		три		<code>раз\&gt;два\&gt;три\&gt;четыре\+\</code>
			четыре	<code>два\+\ три\+\ четыре\</code>
		три		<code>\&lt;три\ четыре\-\-</code>
			четыре	<code>два\-\</code>
		два		<code>раз\&gt;два\&gt;три\&gt;четыре\</code>
раз	два	три	четыре	<code>\end{tabbing}</code>

Описанные в этом разделе возможности окружения `tabbing` бывают полезны (например, для набора стихов), но для печати сложных таблиц в  $\text{\LaTeX}$ 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:  $\text{\TeX}$  предоставляет возможности для печати таблиц, в которых ширина колонок выбирается автоматически (по максимальной ширине их содержимого). В  $\text{\LaTeX}$ 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. Окружение `array` упоминалось гл. II; 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	<code>\begin{tabular}{lc}</code>
<code>itemize</code>	нет	Тип перечня & нумерация <code>\\[5pt]</code>
<code>enumerate</code>	есть	<code>\ttfamily itemize &amp; нет\\</code>
<code>description</code>	нет	<code>\ttfamily enumerate &amp; есть\\</code>
		<code>\ttfamily description &amp; нет\\</code>
		<code>\end{tabular}</code>

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. 109) или окружения `tabbing` (с. 188): 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. Вся таблица, порождаяемая этим окружением, рассматривается Т<sub>Е</sub>X'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  $\text{\LaTeX}$  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. VII); соответственно, на половинной высоте находится и точка отсчета этой «буквы». Пример:

слово	$\begin{array}{cc} A & B \\ B & \Gamma \end{array}$	слово		слово	<pre>\begin{tabular}{rr} A &amp; B\\B &amp; \Gamma \end{tabular}</pre>	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	$\begin{array}{cc} A & B \\ B & \Gamma \end{array}$					слово	<pre>\begin{tabular}{rr} A &amp; B\\B &amp; \Gamma \end{tabular}</pre>					слово	<pre>\begin{tabular}[t]{rr} A &amp; B\\B &amp; \Gamma \end{tabular}</pre>						слово	<pre>\begin{tabular}[b]{rr} A &amp; B\\B &amp; \Gamma \end{tabular}</pre>
-------	---	--	--	--	--	-------	--	--	--	--	--	-------	---	--	--	--	--	--	-------	---

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 линейки задаются с помощью команды `\hline`. Эта команда может следовать либо непосредственно после `\\` (тогда линейка печатается после строки, завершенной этим `\\`), либо непосредственно после `\begin{tabular}` и преамбулы (тогда отрезок печатается перед началом таблицы). Задаваемая командой `\hline` горизонтальная линейка

имеет ширину, равную общей ширине таблицы. Что касается вертикальных линеек, то давайте для начала также ограничимся случаем, когда эти линейки, разделяющие колонки таблицы, простираются на всю ее высоту, сверху донизу. Такие линейки проще всего предусмотреть в преамбуле таблицы. До сих пор мы говорили, что преамбула таблицы — это последовательность из букв `l`, `s` 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
носорог	degunradzis
лев	lauva

```
\begin{tabular}{||l|l||}
\hline
слон & zilonis\\
носорог & degunradzis\\
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.

Таблица V.1.

Популярные напитки			
Название	Старый мельник	Бочкарев	Очаковское
Атрибут	Душевное	Правильное	Живительное
Цена	15		14

## 2.2. Более сложные случаи

**Графы, охватывающие несколько колонок.** Чтобы создать такую, 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 линиями.
- 3) Текст, записываемый в графу.

Пример — в таблице V.1<sup>1</sup>.

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. 197):

```
\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

---

<sup>1</sup>Эта таблица остается в неизменном виде с какого-то из старых изданий. Забавно, наблюдать, насколько за прошедшее время выросли цены.

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. 198 и 202).

**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`, `s` или `r`, описывающей структуру колонки, написать `p{...}`, где вместо многоточия должна быть указана ширина колонки (в  $\TeX$ ’овских единицах — см. с. 19). 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  $\text{\LaTeX}$  понимает слова «одна колонка». Пусть, например, преамбула таблицы имеет вид `||c|1l||r|1l|`, 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	<code>  c 1l  r 1l </code>
Первая колонка	<code>  c </code>
Вторая колонка	<code>1</code>
Третья колонка	<code>1  </code>
Четвертая колонка	<code>r </code>
Пятая колонка	<code>1 </code>
Шестая колонка	<code>1</code>

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}`,

Таблица V.2.

существительные формы		прилагательные формы
мой	$\left\{ \begin{array}{l} \text{le mien, la mienne} \\ \text{les miens, les miennes} \end{array} \right.$	mon, ma, mes
твой	$\left\{ \begin{array}{l} \text{le tien, la tienne} \\ \text{les tiens, les tiennes} \end{array} \right.$	ton, ta, tes
его, ее, свой	$\left\{ \begin{array}{l} \text{le sien, la sienne} \\ \text{les siens, les siennes} \end{array} \right.$	son, sa, ses
наш	le nôtre, la nôtre, les nôtres	notre, nos
ваш	le vôtre, la vôtre, les vôtres	votre, vos
их, свой <sup>1</sup>	le leur, la leur, les leurs	leur, leurs

<sup>1</sup>Лишь в значении принадлежности 3-му лицу.

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 ЛАТ<sub>Э</sub>X'a. 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.

Наш первый пример — таблица французских притяжательных местоимений, взятая из русско-французского словаря акад. Л. В. Щербы (табл. V.2), которую мы задали в ЛАТ<sub>Э</sub>X'e так:

```
\small
\begin{tabular}{c1l}
\multicolumn{2}{c}{существительные формы}
& прилагательные формы\\
\smallskip
мой & $\left\{ \begin{array}{l} \text{le mien, la mienne} \\ \text{les miens, les miennes} \end{array} \right.$ \\
& \end{tabular} \right. $
& mon, ma, mes\\
\bigskip
твой & $\left\{ \begin{array}{l} \text{le tien, la tienne} \\ \text{les tiens, les tiennes} \end{array} \right.$ \\
& \end{tabular} \right. $
& ton, ta, tes\\
\bigskip
его, ее, свой &
```

```

 $\left\{ \begin{array}{l} \text{le sien, la sienne} \\ \text{les siens, les siennes} \end{array} \right.$ 
& 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}{Лишь в значении принадлежности 3-му лицу.} \\
\end{array}

```

Разберем, как устроена эта таблица. Как явствует из ее преамбулы `c1l`, она состоит из трех колонок, из которых левая центрирована, а две другие прижаты («выключены») 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,



Таблица V.3.

Понедельник	8 <sup>30</sup> –15	Обед	11–12
Вторник	12–19	Обед	15–16
Среда	10–17	Обед	12 <sup>30</sup> –13 <sup>15</sup>
Четверг	9–17	Обед	12–13
Пятница	11–16	Обед	–
Суббота	8–14	Обед	11–12

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).

Следующий пример (табл. V.3) — расписание работы одной химчистки. Для него исходный текст выглядит так:

```
\begin{tabular}{lr@{--}l@{\quad}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, что 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»<sup>2</sup>: символ @, непосредственно после которого в фигурных скобках записан какой-то текст, возможно, с Т<sub>Е</sub>X'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 `\qqquad` и `\quad` вокруг этого слова? Дело в том, что между колонками, разделенными at-выражением, *не* вставляется дополнительный интервал, которым Л<sup>A</sup>T<sub>Е</sub>X разделяет колонки в 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`.

---

<sup>2</sup>Мы выбрали для него такое название, поскольку официально символ @ называется «коммерческое at»; неофициально этот символ называют самыми разными именами, от «собаки» до «блямбы».

Теперь разберем обещанный пример, в котором приходится менять заданное по умолчанию расстояние между колонками. Посмотрите на такую формулу:

$$\begin{array}{r|l} x^2 + 2x - 12 & x + 5 \\ \hline x^2 + 5x & x - 3 \\ - 3x - 12 & \\ - 3x - 15 & \\ \hline & 3 \end{array}$$

Она была создана с помощью следующих L<sup>A</sup>T<sub>E</sub>X'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. 197), и в результате третья строка закончилась бы вертикальным отрезком, что нам совсем ни к чему.

Осталось заметить, что знаки `\[` и `\]`, 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. 194: 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  $\text{\LaTeX}$  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  $\text{\LaTeX}$ 'у не помогла: ведь `\strut` все равно выше! Чтобы обойти эту трудность, в  $\text{\LaTeX}$ '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. VI, 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»  $\text{\LaTeX}$ 'у (без подключения дополнительных стилевых пакетов). Ряд стилевых пакетов, входящих в комплект поставки  $\text{\LaTeX}$ '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»  $\text{\LaTeX}$ '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. 19); по умолчанию его величина равна нулю, для лучшего отделения линеек от текста хорошо присвоить ему значение 2–3 пункта.

а	б	в	г					<code>\begin{tabular}[t]{rrrr}</code>
д	е	ж	з					<code>a &amp; б &amp; в &amp; г \\</code>
				а	б	в	г	<code>д &amp; е &amp; ж &amp; з</code>
				д	е	ж	з	<code>\end{tabular}</code>
								<code>\begin{tabular}[t]{ rrrr }</code>
								<code>\hline a &amp; б &amp; в &amp; г \\</code>
								<code>д &amp; е &amp; ж &amp; з \\ \hline</code>
								<code>\end{tabular} \\[1in]</code>
								<code>\begin{tabular}[t]{rrrr}</code>
								<code>a &amp; б &amp; в &amp; г \\</code>
								<code>д &amp; е &amp; ж &amp; з</code>
								<code>\end{tabular}</code>
								<code>\begin{tabular}[t]{ rrrr }</code>
								<code>\firsthline a &amp; б &amp; в &amp; г \\</code>
								<code>д &amp; е &amp; ж &amp; з \\ \hline</code>
								<code>\end{tabular}</code>

Рис. V.1.

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			<code>\begin{tabular}[t]{rr}</code>
в	г	а	б	<code>a &amp; б \\ \v &amp; г</code>
		в	г	<code>\end{tabular}</code>
				<code>\begin{tabular}[t]{ rr }</code>
				<code>\hline a &amp; б \\ \v &amp; г \\</code>
				<code>\hline</code>
				<code>\end{tabular}</code>

Чтобы выравнивание происходило не по линейке, а по первой строке текста, надо задать верхнюю линейку командой `\firsthline`, а не `\hline`, как видно из рис. V.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.

Настроение бодрое, идем ко дну.	Настроение бодрое, идем ко дну.	Настроение бодрое, идем ко дну.
---------------------------------	---------------------------------	---------------------------------

```
\begin{tabular}{b{1.2in}m{1.2in}p{1.2in}}
```

Настроение бодрое, идем ко дну. &
Настроение бодрое, идем ко дну. &
Настроение бодрое, идем ко дну.\\

```
\end{tabular}
```

Рис. V.2.

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. V.2.

Наряду с `at`-выражениями, пакет `array` позволяет использовать в преамбуле еще и `!`-выражения. Именно, между обозначениями для колонок 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 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. 136 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	<code>\begin{tabular}{ &gt;{Генрих }l &gt;{\slshape}l }</code>
Генрих III	1551–1589	<code>\hline</code>
Генрих IV	1553–1610	<code>II &amp; 1519--1559\III &amp; 1551--1589\</code>
		<code>IV &amp; 1553--1610\ \hline</code>
		<code>\end{tabular}</code>

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	$(x + y)^2$	<code>\begin{tabular}{ l&gt;{\$}l&lt;{\$} }</code>
квадрат разности	$(x - y)^2$	<code>квадрат суммы &amp; (x+y)^2\</code>
сумма кубов	$x^3 + y^3$	<code>квадрат разности &amp; (x-y)^2\</code>
		<code>сумма кубов &amp; x^3+y^3</code>
		<code>\end{tabular}</code>

## 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. 194). Если подключить стилиевой 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:

A	B	B	Г
Д	Е	Ж	З

```

\begin{tabular}{|c|cc|c|}
\hline A & 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	$\vdash$	$\dashv$	$\vdash, \top, \perp$	$\vDash$	$\vDash$	$\#, \top, \perp$
В аргументе <code>\hhline</code>	<code>  -</code>	<code>-  </code>	<code>-   -</code>	<code>  =</code>	<code>=  </code>	<code>=   =</code>
На печати	$\vDash$	$\vDash$	$\#, \top, \perp$	$\vDash$	$\vDash$	$\vDash$
В аргументе <code>\hhline</code>	<code>:=</code>	<code>=:</code>	<code>:= =</code>	<code>   -</code>	<code>-   </code>	<code>-    -</code>
На печати	$\vDash$	$\vDash$	$\vDash$	$\#$	$\#$	$\#, \top, \perp$
В аргументе <code>\hhline</code>	<code>  :=</code>	<code>:=  </code>	<code>:= :=</code>	<code># =</code>	<code>= #</code>	<code># = =</code>
На печати	$\ulcorner$	$\llcorner$	$\lrcorner$	$\lrcorner$	$\top$	$\perp$
В аргументе <code>\hhline</code>	<code>  t :=</code>	<code>  b :=</code>	<code>= : t  </code>	<code>= : b  </code>	<code>= : t :=</code>	<code>= : b :=</code>

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 ЛАТЭХ'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. Вот пример того, что может получиться.

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.
Винди	кобель	уайттерьер	отл.
Грант	кобель	ризеншнауцер	отл.
Денни	кобель	кокер-спаниель	оч. хор.
Джек	кобель	дог	отл.
Джерри	кобель	ирландский терьер	отл.
Ероха	кобель	метис	отл.
Жорж	кобель	бриар	оч. хор.
Зента	сука	ирландский терьер	оч. хор.
Зоя	сука	ризеншнауцер	отл.
Клия	сука	бладхаунд	отл.
Малыш	кобель	метис	отл.
Найт	кобель	ризеншнауцер	отл.
Одри	сука	ризеншнауцер	отл.
Ральф	кобель	чихуахуа	отл.
Ричард	кобель	ирландский сеттер	отл.
Рэм	кобель	эрдельтерьер	оч. хор.
Сэнди	кобель	немецкая овчарка	отл.
Тима	кобель	миттельшнауцер	отл.
Чапик	кобель	метис	отл.

*Продолжение на следующей странице*

1	2	3	4
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 & миттельшнацер & отл.\\
...
Эмир & кобель & эрдельтерьер & оч. хор.\\
\end{longtable}
```

Опишем теперь отличия окружения `longtable` от `tabular`.

Во-первых, после первого запуска  $\text{\LaTeX}$ 'а колонки таблицы, определенной как `longtable`, могут оказаться невыровненными (это связано с тем, что при первом проходе  $\text{\LaTeX}$  читает такую таблицу не целиком, а по кускам, и выравнивает эти куски независимо друг от друга). Чтобы добиться выравнивания, надо запустить  $\text{\LaTeX}$  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`.

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 в ширину. 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 VI

## Создание новых команд

Средства  $\text{\LaTeX}$ '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  $\text{\TeX}$ '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  $\stackrel{\text{def}}{=}$  (он означает «равно по определению»). Пользуясь тем, что вы узнали из гл. II, нетрудно понять, что его можно сгенерировать такой последовательностью команд:

```
\stackrel{\mathrm{def}}{=}
```

Часто писать такой длинный набор значков утомительно. Вот бы в  $\text{\LaTeX}$ 'е была предусмотрена команда, скажем, `\eqdef`, генерирующая символ бинарного отношения  $\stackrel{\text{def}}{=}$ ! Правда, такой команды нет, но мы ее можем создать. Для этого следует написать так:

```
\newcommand{\eqdef}{\stackrel{\mathrm{def}}{=}}
```

После того как  $\text{\TeX}$  прочтет эту строку, он всюду, встречая команду `\eqdef`, будет реагировать точно так же, как если бы он видел текст `\stackrel{\mathrm{def}}{=}`. Например, формула  $x^2 \stackrel{\text{def}}{=} x \cdot x$  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 латинских букв (прописные и строчные различаются, русские буквы в многобуквенных именах макросов использовать нельзя). 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 попытаетесь 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<sup>1</sup>, 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.

Имя новой команды не должно начинаться на `end`. Кроме того, 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.

Макросы хороши как средство скорописи. Например, если в вашем тексте часто встречается знак  $\Delta$ , то вам может надоесть все время

---

<sup>1</sup>Фигурные скобки, входящие в состав команд `\{` и `\}`, в счет при этом не идут.

писать длинную команду `\bigtriangleup`. Коли так, придумайте сокращенное обозначение (скажем, `\btu`), напишите в преамбуле

```
\newcommand{\btu}{\bigtriangleup}
```

и вы сможете писать формулы наподобие

$$(A \triangle B) \cap C = (A \cap C) \triangle (B \cap C) \quad \begin{array}{l} \$ (A \backslash btu B) \backslash cap C = \\ (A \backslash cap C) \backslash btu (B \backslash cap C) \$ \end{array}$$

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  $\mathbf{P}^n$  --- 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 разворачивании 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  $\mathbb{P}^n$  —  $n$ -мерное проективное пространство,  
а  $X \subset \mathbb{P}^n$  — неприводимое многообразие...

(знаки  $\sim$  мы поставили, чтобы строчка не смогла начаться с тире — с. 91). Команда `\ensuremath` *всегда* обрабатывает свой аргумент как математическую формулу, независимо от того, в тексте или в формуле вы ее используете.

Даже и в таком виде определенная нами команда `\Pn` обладает одним недостатком: в ситуациях наподобие

Трехмерные подмногообразия в  $\mathbb{P}^n$  называются...

нам приходится специально организовывать пробел после  $\mathbf{P}^n$  с помощью команды «backslash с пробелом», чтобы формула не слилась со следующим словом. Ничего страшного в этом нет, но если вы хотите и об этом не заботиться, загляните в разд. 1.5 ниже.

Создавать макросы полезно не только для сокращения числа нажатий на клавиши при наборе формул. Вот пример, когда макросы помогают при наборе обычного текста. Предположим, в нашем тексте много задач, причем условие каждой из задач начинается 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:

**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 разворачивании 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 разворачивании 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. 125):

```
\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. 125 и ниже), а если ее не будет, то `\par` сыграет роль недостающей пустой строки.

Возможно, вам (или вашему редактору) захочется, чтобы перед очередной задачей был вертикальный отступ (полиграфист сказал бы «отбивка»). Пожалуйста, нам нетрудно:

```
\newcommand{\z}{\par\medskip
\textbf{Задача. }}
```

Впрочем, `\medskip` в этом месте — не самое лучшее решение. Когда вы пишете макрос, вы не можете заранее знать, в какой ситуации он будет употреблен: если перед вашей задачей уже имеется другой вертикальный отступ, он просуммируется с вашим `\medskip`’ом, в результате чего отступ перед задачей окажется больше, чем надо. Грамотный Т<sub>Е</sub>Xник в этом месте напишет так:

```
\newcommand{\z}{\par\addvspace{\medskipamount}
\textbf{Задача. }}
```

(пробел после `\addvspace`, создаваемый концом строки, ни на что не влияет: в момент обработки этой команды Т<sub>Е</sub>X находится в вертикальном режиме, в котором пробелы игнорируются). Команда `\addvspace`, подобно известной нам `\vspace`, добавляет вертикальные отступы, но делает это более хитрым образом. Именно, если перед ней вертикального отступа не было<sup>2</sup>, то она действует так же, как `\vspace`, если был вертикальный отступ, не превосходящий указанного в ее аргументе, то она увеличивает его до указанного в ее аргументе размера, а если был вертикальный отступ, превышающий размер из ее аргумента, то она вообще ничего не делает.

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\addvspace{\medskipamount}
\noindent\textbf{Задача. }}
```

Если бы каждую задачу мы оформляли вручную, нам пришлось бы перед каждым словом «**Задача**» вписывать команду `\noindent`. В дальнейшем, познакомившись с Л<sup>A</sup>T<sub>Е</sub>X’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 «Лекция», а не «Глава», то необходимо определить по-новому команду `\chaptername`. Для такого

---

<sup>2</sup>Имеется в виду не пустая строка в `tex`-файле, а вертикальный отступ в тексте, генерируемом при обработке `tex`-файла.

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  $\alpha$  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 ЛАТЭХ'a определяются в терминах других команд ЛАТЭХ'a, причем знать эти сложные зависимости 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 ТЭХ'овскими командами почти нет. В частности, почти все команды ЛАТЭХ'a, 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 ТЭХ'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» ТЭХ'a. 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 часто возникает потребность создать новую команду, принимающую аргументы. Пусть, например, в вашем тексте часто встречается «символ Лежандра», выглядящий так:  $\left(\frac{a}{l}\right)$ . Для получе-

ния 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, L<sup>A</sup>T<sub>E</sub>X предоставляет нам возможность сделать и это. Для создания команды с аргументами используется все та же команда `\newcommand`, но с необязательным аргументом<sup>3</sup>. 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 самом «замещающем тексте» появились значки `#1` и `#2`. При разворачивании макроса на их место будут подставляться соответственно первый и второй аргументы нашей новой команды `\smb`. Например, если в формуле написать

```
\smb{a+b}{c}
```

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 разворачивании 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 L<sup>A</sup>T<sub>E</sub>X'овской команды). При разворачивании 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.

Вот пример. Если определить команду `\shuffle` как

```
\newcommand{\shuffle}[4]{K#4к#2#1л}
```

---

<sup>3</sup>Можно (и разумно) использовать в этой ситуации команду `\newcommand*` («вариант со звездочкой») См. по этому поводу с. 222 ниже.

тогда будет получаться, например, такое:

Крокодил Гена и его друзья.	<code>\shuffle{ди}{о}{го}{по}</code>
	Гена и его друзья.

В самом деле, первым аргументом команды `\shuffle` будет `ди`, вторым `о`, третьим `го`, четвертым `по`; при разворачивании 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 ЛАТЭХ'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`!)

### 1.3. Команда `\newcommand` со звездочкой

У команд `\newcommand` и `\renewcommand` существуют варианты «со звездочкой» (см. с. 18), называемые `\newcommand*` и `\renewcommand*`. Они работают точно так же, как их тезки без звездочек, со следующим отличием: если команда с аргументами определялась с помощью `\newcommand*` или `\renewcommand*`, то в ее аргументе *не может содержаться пустая строка или команда `\par`*<sup>4</sup>. 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, как в следующем примере (забыта фигурная скобка в самой первой формуле; команду `\smb` мы определили в учебных целях на с. 220):

Символ Лежандра  $\$ \text{\smb{a}} \{1\} \$$ , где  $\$a\$$  не делится на  $\sim 1 \$$ , равен по определению  $\sim 1 \$$ , если  $\$a\$$  является квадратичным вычетом по модулю  $\sim 1 \$$ , и  $\sim -1 \$$  в противном случае.

Вопрос о том, как зависит  $\$ \text{\smb{a}} \{1\} \$$  от  $\$1 \$$  при фиксированном  $\sim \$a \$$ , 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. 220), то  $\text{\TeX}$  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  $\text{\TeX}$  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)}
```

---

<sup>4</sup>На содержимое самого «замещающего текста» (второго обязательного аргумента `\newcommand*`) никаких дополнительных запретов не накладывается.

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{1}\$, где  $a$  не делится на  $1$ , равен  $n$  \ETC.

! Paragraph ended before \smb was complete.

<to be read again>

\par

1.8

?

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...).

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.

## 1.4. Создание команд с необязательным аргументом

Мы уже привыкли, что многие  $\LaTeX$ ’овские команды допускают необязательный аргумент, причем в самых разных вариантах: иногда он один, иногда их несколько, иногда он ставится перед обязательными, иногда после, а бывает еще, что один до, а другой после... Средствами, доступными рядовому пользователю  $\LaTeX$ ’а, всего этого разнообразия так просто не воспроизвести, но все же имеется возможность создать собственную команду с одним необязательным аргументом, который должен ставиться перед всеми обязательными. Такая конструкция имеет смысл, если один из аргументов вашего макроса в большинстве случаев будет принимать одно и то же значение (назовем его «значением по умолчанию»). Тогда можно действовать следующим образом: аргумент, имеющий «значение по умолчанию», объявить аргументом номер один (тем самым в «замещающем тексте» макроса вместо него надо будет писать `#1`), а в определении макроса поместить это значение по умолчанию, в дополнительной паре квадратных скобок, между квадратными скобками, в которых указано количество аргументов, и фигурными скобками с замещающим текстом.



Более формально: если вы хотите определить команду с  $n$  обязательными аргументами и одним необязательным, имеющим «значение по умолчанию»  $D$ , то надо написать так:

```
\newcommand{имя_команды}[n+1][D]{замещающий текст}
```

(и аналогично для `\renewcommand` и их вариантов со звездочкой).

Вот пример использования этой конструкции. В математических формулах часто встречаются перечисления вида  $x_1, \dots, x_n$  или, например,  $\alpha_1, \dots, \alpha_{2n+1}$ . В какой-то момент автор решил сократить число нажатий на клавиши и определил макрос

```
\newcommand*{\lst}[2]{#1_1,\ldots,#1_{#2}}
```

после чего первую из формул стало возможным записать в формуле как `\lst xn` (вспомним, что когда одиночная буква передается в качестве аргумента, в фигурные скобки ее можно не брать), а вторую, соответственно, как `\lst\alpha{2n+1}`. Иногда, однако же, такие перечисления начинаются не с единицы; хочется предусмотреть в макросе и это, но не хочется всякий раз, когда перечисление начинается с единицы (то есть в большинстве случаев) эту единицу явно выписывать. Вот тут и пригодится макрос с необязательным аргументом! Если написать

```
\newcommand*{\lst}[3][1]{#2_{#1}, \ldots, #2_{#3}}
```

(здесь `[3]` означает общее количество аргументов, включая необязательный `#1`, а `[1]` — то, что по умолчанию необязательный аргумент равен 1), то  $x_1, \dots, x_n$  в формуле можно будет по-прежнему записывать как `\lst xn`, но  $x_{n+1}, \dots, x_{2n}$  можно будет записать в формуле как `\lst[n+1]x{2n}`. Обратите внимание, что смысл знаков `#1` и `#2` в замещающем тексте теперь изменился: необязательный аргумент при таком определении обязательно должен иметь номер один.

В определении, которым я реально пользуюсь, предусмотрены еще два усовершенствования. Во-первых, весь замещающий текст помещен дополнительно в аргумент команды `\ensuremath` (нередко такого рода перечисления составляют отдельную формулу, и писать лишние доллары не хочется), во-вторых, я всегда подключаю пакет `xspace`, и в самом конце этого определения у меня стоит команда `\xspace` — в следующем разделе объясняется, что это такое и зачем это нужно.

### 1.5. Для эстетов: пакет `xspace`

Если определить команду без аргументов, которую предполагается в дальнейшем употреблять не в формулах, где все пробелы все равно игнорируются, но в тексте, то, как мы знаем с самой первой главы (с. 13),

пробел после имени команды, сигнализирующий интерпретатору  $\text{\TeX}$ ’а о том, что имя команды закончилось, на печати никак не отражается, и если на печати пробел все же необходим, то его надо специально организовывать. Это не всегда удобно. Пусть, например, мы определили макрос

```
\newcommand{\R}{\ensuremath{\mathbb R}}
```

(стандартное обозначение для поля вещественных чисел). Благодаря `\ensuremath` это `\R` можно употреблять и в формулах, и в тексте, но при этом в тексте приходится писать что-нибудь вроде

Поле `\R` является вещественно замкнутым

(пробел приходится создавать самому, чтобы  $\mathbb{R}$  не сливалось со словом «вещественно»). Если «зашить» этот дополнительный пробел в макроопределение

```
\newcommand{\R}{\ensuremath{\mathbb R}\ }
```

то плохо будет уже с формулами: в формуле `\R^3` на печати появится нелепый пробел между  $\mathbb{R}$  и верхним индексом (да и с текстом не все будет гладко: пробел, прописанный нами в макросе, будет возникать и в случае, когда сразу после  $\mathbb{R}$  стоит знак препинания, или в сочетаниях типа  $\mathbb{R}$ -линейный, где он никому не нужен).

Пакет `xspace` позволяет следующим образом избежать этих неприятностей. После того, как он подключен, определим команду `\R` следующим образом:

```
\newcommand{\R}{\ensuremath{\mathbb R}\xspace}
```

После этого все проблемы исчезнут:

Поле  $\mathbb{R}$ . Поле  $\mathbb{R}$  и пространство  $\mathbb{R}^3$ . Поле `\R`.

Поле `\R` и пространство `\R^3`.

Команда `\xspace`, благодаря которой все это происходит, действует как «умный пробел»: если за ней следует просто буква, она разворачивается в пробел, а если знак препинания (или, скажем, команда `\footnote`), то в пустоту. Стало быть, внутри формул, где все пробелы игнорируются, эта команда практически равносильна команде «ничего не делать», а в тексте пробел получается ровно там, где нужно.

Как водится, компьютерный ум имеет свои пределы: иногда может случиться, что `\xspace` (точнее, макрос, определение которого заканчивается на `\xspace`) создаст-таки лишний пробел. Чтобы от него избавиться, надо после этого макроса поставить пару из открывающей и закрывающей фигурных скобок `{}` — тогда нежелательного пробела не возникнет. Вот пример (с неизбежностью глупый).

```
\newcommand{\ghnm}{>>} %% Зачем???
<<В поле \R{\ghnm
```

Если опустить {}, то между  $\mathbb{R}$  и » возникнет лишний пробел.

Если после `\xspace` стоит «backslash с пробелом», то дополнительного пробела она не создаст.

## 2. Счетчики

В этом разделе мы научимся самостоятельно организовывать автоматическую нумерацию, подобно тому, как  $\text{\LaTeX}$  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.

### 2.1. Создание счетчиков и простейшие операции с ними

Каждый  $\text{\LaTeX}$ ’овский счетчик имеет свое имя — последовательность латинских букв (без знака \ ). Прописные и строчные буквы в именах счетчиков различаются. 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  $\text{\LaTeX}$ ’е уже определены в момент начала обработки вашего текста), то  $\text{\LaTeX}$  создавать счетчик с таким именем откажется и выдаст сообщение об ошибке (имя счетчика может совпадать с именем уже существующей команды: учет имен счетчиков и имен команд ведется независимо.).

В отличие от многих других  $\text{\LaTeX}$ ’овских команд, команда для создания нового счетчика является «глобальной»: даже если она давалась внутри группы,  $\text{\LaTeX}$  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, использу-

ется 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.

Можно прибавить к счетчику какое-нибудь целое число, для чего 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`:

40 мальчиков подряд.

```
\setcounter{abcd}{40}  
\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):

Людовик XIV

```
\setcounter{abcd}{14}  
Людовик \Roman{abcd}
```

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):

На седьмом месте в латинском алфавите стоит буква g.

```
\setcounter{abcd}{7}
На седьмом месте
в латинском алфавите
стоит буква \alph{abcd}.
```

Если значение счетчика при использовании этими командами превышает количество букв в латинском алфавите, то  $\text{\LaTeX}$  выдает сообщение об ошибке.

Чтобы иметь возможность напечатать *русскую* букву с номером, равным значению счетчика, русский текст должен быть должным образом оформлен (это означает, что  $\text{\TeX}$  должен иметь возможность воспринимать русские буквы во входном файле, а также что должен быть подключен пакет `babel` с опцией `russian`; см. разд. IV.1 и приложение II). Если это сделано, то для таких целей можно использовать команду `\asbuk` для строчных букв и `\Asbuk` для прописных.

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: \*, †, ‡, 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.

Применим наши познания к делу. На с. 218 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 А...
```

и при этом знать, что номера  $\text{\LaTeX}$  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  $\backslash z$  надо предусмотреть, чтобы всякий раз значение этого счетчика увеличивалось на единицу, а затем печаталось в качестве номера задачи. В качестве имени счетчика выберем бесхитрое «*zadacha*»:

```
 $\backslash newcounter\{zadacha\}$ 
```

(напомним, что при выполнении этой команды счетчику *zadacha* будет присвоено значение 0). Теперь модифицируем определение макроса  $\backslash z$  так:

```
 $\backslash newcommand\{z\}\{\par\addvspace\{\medskipamount\}$   
 $\backslash addtocounter\{zadacha\}\{1\}\%$   
 $\backslash textbf\{Задача \arabic\{zadacha\}\. \}$ 
```

Напомним, что команда  $\backslash par$  означает «завершить предыдущий абзац, если он еще не был завершен»; без нее пришлось бы следить за тем, чтобы команда  $\backslash z$  ставилась только после пустой строки. Знак процента мы поставили, чтобы убрать лишний пробел, порождаемый концом строки. Теперь при первом исполнении команды  $\backslash z$  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!

## 2.2. Отношение подчинения между счетчиками

Команда  $\backslash z$ , 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  $\text{\LaTeX}$ '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  $\backslash section$  значение счетчика *section* увеличивается на 1, и значение этого счетчика в каждый момент равно номеру текущего раздела. Поэтому, если в определении команды  $\backslash z$  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\addvspace{\medskipamount}
\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.

Точные правила создания счетчика, подчиненного другому счетчику, 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 уточнить, 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 L<sup>A</sup>T<sub>E</sub>X'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.



### 2.3. 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\addvspace{\medskipamount}
\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 ЛАТЭХ 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 ЛАТЭХ'овской конструкцией, связанной со счетчиками.

Мы уже знаем, что значение ЛАТЭХ'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:

```
Людовик XIV \renewcommand{\theabcd}%
              {\Roman{abcd}}
              \setcounter{abcd}{14}
              Людовик \theabcd
```

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.

## 2.4. Счетчики, которые уже определены

Мы уже мельком упоминали, что при начале работы ЛАТ<sub>Е</sub>X'a некоторые счетчики определены сразу. Например, это те счетчики, которые перечислены на с. 231. 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. VIII.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}}
```

## 2.5. Счетчики и перечни

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. 49). Если окружение `itemize` расположено внутри другого окружения `itemize`, как в примере на с. 117, 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.

Другой вариант — вообще не связываться в этом месте с математическим режимом: вместо этого можно подключить пакет `textcomp` (см. разд. III.1.4) и задать искомый символ командой `\textsurd`.

Теперь рассмотрим окружение `enumerate`. Коль скоро оно автоматически нумерует элементы перечня, можно предположить, что это окружение связано с ЛАТЭХ'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 `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.

### 3. Окружения типа «теорема»

Если вы пишете математический текст, то в нем будет содержаться небольшое 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 Т<sub>E</sub>X'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). Удобнее всего соответствующие макросы (точнее говоря, новые окружения) создавать из полуфабрикатов, предоставляемых нам для этих целей Л<sup>A</sup>T<sub>E</sub>X'ом.

Окружения, используемые в Л<sup>A</sup>T<sub>E</sub>X'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 Л<sup>A</sup>T<sub>E</sub>X в целях экономии машинной памяти и исходя из того, что на все вкусы в любом случае не напасешься, определять их предоставляет вам. Как это делать, удобнее всего разобрать на примере.

Пусть в нашем тексте присутствуют «предложения». Давайте создадим окружение `pred1` таким образом, чтобы можно было, например, писать

<b>Предложение 1.</b> <i>Волга впадает в Каспийское море.</i>	<code>\begin{pred1}</code> Волга впадает в Каспийское море. <code>\end{pred1}</code>
<b>Доказательство.</b> См. любую географическую карту.	<code>\textbf{Доказательство.}</code> См. любую географическую карту.

Для создания такого окружения используется команда `\newtheorem`: надо написать в преамбуле

```
\newtheorem{pred1}{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 стилевыми пакетами, предоставляемыми Американским математическим обществом, появляются дополнительные возможности влиять на оформление «теорем». См. ниже разд. VI.3.1.

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. Это не всегда удобно: часто хочется, например, в чтоб 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 3.1.** *Сумма углов треугольника равна  $180^\circ$ .*

```
\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,  $\text{\LaTeX}$  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.

### 3.1. Окружения типа «теорема» в пакете `amsthm`

Все  $\text{\LaTeX}$ '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 (и входящий во все современные поставки  $\text{\TeX}$ 'а), позволяет внести в это оформление некоторое разнообразие. Итак, предположим, что вы его подключили. Что нового, по сравнению с «чистым»  $\text{\LaTeX}$ '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, предложений и лемм, `definition` — для определений, `remark` — для замечаний<sup>5</sup>. 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  $\square$ . Если вас не устраивает, что слово «доказательство» пишется по-английски, нужно переопределить с помощью `\renewcommand` команду `\proofname` (ср. с. 153). Если символ  $\square$  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.

## 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 предыдущих разделах мы, разрабатывая команду `\z`, в явном виде задали вертикальный отступ перед очередной задачей. Если этот отступ захочется изменить, то придется снова лезть в определение команды `\z`. Было бы удобнее, если бы в нашем распоряжении был параметр под названием, скажем, `\otstup`, так что можно было бы в определении команды `\z` написать

```
\vspace{.5em plus 2pt minus 1pt}
```

---

<sup>5</sup>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.



и потом отдельно написать, допустим,

```
\otstup=.5em plus 2pt minus 1pt
```

Правда, такого параметра нет, но 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.

Команда `\newlength` имеет один обязательный аргумент — имя команды, обозначающей определяемый вами параметр. Это имя должно подчиняться обычным правилам для  $\text{\TeX}$ -овских команд (backslash, после которого следует либо одна не-буква, либо последовательность букв). Если это имя уже занято,  $\text{\LaTeX}$  выдаст сообщение об ошибке. Определение нового параметра, совершаемое командой `\newlength`, является «глобальным»: даже если эта команда была дана внутри группы,  $\text{\TeX}$  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  $\text{\TeX}$ -овские и  $\text{\LaTeX}$ -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  $\text{\TeX}$ -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  $\text{\TeX}$ -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  $\text{\LaTeX}$ -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. 130; 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                                \primer=10mm
8      8                                9\hspace{\primer}9
9      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, L<sup>A</sup>T<sub>E</sub>X предоставляет полезную команду

```
\settowidth{параметр}{текст}
```

которая присваивает *параметру* значение, равное ширине *текста*. Вот пример:

```
СЛОВО слово           \settowidth{\primer}{\Large
                        слово }
                        {\Large слово }слово

                        \hspace{\primer}слово
```

(кстати, без помощи `tabbing` или `tabular` получилось выравнивание).

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. VII.1). `\settodepth` присваивает *параметру* значение, равное максимальному расстоянию, на которое *текст* опускается ниже «базисной линии» (см. с. 246).

В разд. III.9.4 у нас шла речь о том, что некоторые используемые в T<sub>E</sub>X'e длины 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: общий случай

Использование команды `\newtheorem` — частный случай задачи определения нового окружения. Новые окружения есть смысл определять, когда для достижения необходимого нам эффекта требуется сложная последовательность команд в начале и в конце какого-то текста. Вот как определяют новые окружения в общем случае.

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, предназначен- ное для верстки таблиц, можно исполь- зовать 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 определения окружения 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...
\конец
```

Преимущество оформления такого рода конструкций в виде окружений состоит в том, что при этом легче контролировать ошибки: если вы напишете `\begin{рамка}` и при этом забудете написать соответствующую команду `\end{рамка}`, то  $\text{\LaTeX}$  выдаст сообщение об ошибке, в котором именно это вам и скажет; если же вы забудете команду `\конец`, 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 `рамка` таким образом, чтобы ширина текста в рамке передавалась ему как аргумент. Соответствующее определение будет выглядеть так:

```
\newenvironment{рамка}[1]{\begin{tabular}{|p{#1}|}
\hline}{\hline\end{tabular}}
```

После этого можно писать, например,

```
\begin{рамка}{6cm}
Текст...
\end{рамка}
```

или даже

```
\begin{рамка}{.85\textwidth}
Текст...
\end{рамка}
```

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, \dots, #9`, причем эти значки можно употреблять *только в открывающих командах* (т.е. во втором обязательном

аргументе команды `\newenvironment`). Можно также определить окружение, принимающее один необязательный аргумент — это делается по тем же правилам, что и определение команды с одним необязательным аргументом (см. разд. 1.4).

С помощью `\newenvironment` нельзя переопределить уже существующее окружение (если вы все же попытаете так сделать,  $\text{\LaTeX}$  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`.

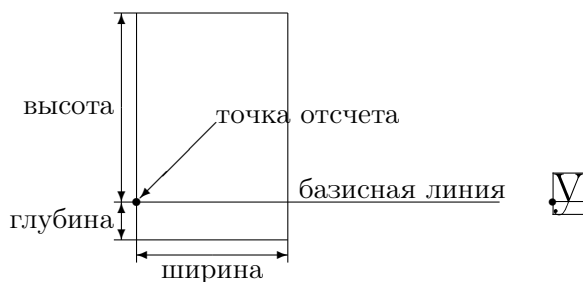
# Глава VII

## Блоки и клей

### 1. Текст состоит из блоков

Мы уже отмечали, что в процессе набора  $\text{\TeX}$  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.

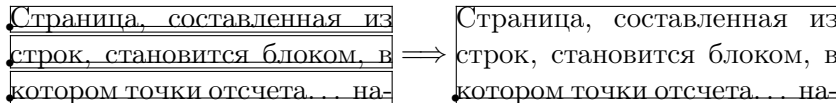
С точки зрения  $\text{\TeX}$ 'а, каждая буква представляет собой *блок* (английский термин: *box*), т. е. прямоугольник с выделенной *точкой отсчета*; горизонтальная прямая, проходящая через точку отсчета, называется *базисной линией* (английский термин: *baseline*). Блок характеризуется тремя размерами: шириной, высотой и глубиной. См. рисунок, на котором также изображен блок, соответствующий букве *y*.



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:

Крокодил  $\Rightarrow$  Крокодил

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:



В приведенных выше примерах мы сталкивались с блоками, которые  $\text{\TeX}$  создает автоматически; в настоящей главе пойдет речь о командах, предназначенных для создания блоков вручную. Сначала мы расскажем, какие средства для этого предоставляет нам  $\text{\LaTeX}$ , а затем рассмотрим некоторые  $\text{\TeX}$ 'овские команды, дающие дополнительные возможности.

## 2. Команды $\text{\LaTeX}$ 'а для генерации блоков

### 2.1. Блоки из строк

С одной командой для генерации блоков мы уже знакомы: это команда  $\text{\mbox}$ . Эта команда создает блок из текста, набираемого в одну строку. Полученный блок рассматривается  $\text{\TeX}$ 'ом как одна буква:

Проказница мартышка,  
осел, козел и косола-  
пый мишка...

Проказница мартышка,  
 $\text{\mbox}{осел, козел}$  и  
косолапый мишка\ldots

В этом примере  $\text{\TeX}$  никогда не разорвет строку между словами «осел» и «козел» и никогда не сделает переносов в этих словах: при верстке абзаца  $\text{\TeX}$  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  $\text{\TeX}$  не сможет растянуть или сжать пробел между словами «осел» и «козел» для выравнивания строк в абзаце.

Теперь, когда мы знаем, что такое  $\text{\TeX}$ 'овские блоки, можно признать, что окружения  $\text{\tabular}$  и  $\text{\array}$  тоже генерируют блоки, и именно поэтому создаваемый ими текст воспринимается  $\text{\TeX}$ 'ом как одна большая буква.

В аргументе команды  $\text{\mbox}$  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  $\text{\mbox}$ , или даже окружение  $\text{\tabular}$ ) и т. д. Запрещены в аргументе команды  $\text{\mbox}$  пустые строки или команды  $\text{\par}$ , выключающие 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 Т<sub>Е</sub>X'овских единиц, либо как параметр со значением длины, возможно — с числовым коэффициентом (см. разд. VI.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 Т<sub>Е</sub>X'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 Т<sub>Е</sub>X'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	<code>\parindent=0pt</code>
	екст	<code>\makebox[10em][r]{текст}\</code>
	кст	<code>\makebox[10em][r]{екст}\</code>
	текст	<code>\makebox[10em][r]{кст}\</code>
текст		<code>\makebox[10em][c]{текст}\</code>
		<code>\makebox[10em][l]{текст}</code>

Мы установили нулевое значение абзацного отступа, чтобы все строки, включая первую, начинались с самого начала.

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 <del>текст</del>	<code>текст\makebox[0pt][l]{???}текст\</code>
текст <del>текст</del>	<code>текст\makebox[0pt][r]{???}текст\</code>

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 `\makebox` в явном виде. Можно, кроме того, выразить эту ширину через «естественную» ширину текста. Для этого служит команда `\width`. Вот, например, как сделать, 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  $\text{\TeX}$ -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

```

        вставили целый абзац
        текста, сверстанного
В строку по всем  $\text{\TeX}$ -овским      прерванная строка.
        правилам. После этого
        продолжается

```

получился следующим образом:

```

В строку\qquad
\parbox{4cm}{вставили целый
абзац текста, сверстанного
по всем  $\text{\TeX}$ -овским правилам.
После этого продолжается}\qquad
прерванная строка.

```

Как видите, базисная линия блока, создаваемого командой `\parbox`, находится в точности посередине текста. Поэтому команду `\parbox` удобно использовать для включения больших фрагментов текста в математические формулы. Например, формула

$$\int_a^b f'(x) dx = f(b) - f(a)$$

для всех функций  $f$ ,  
производная которых  
интегрируема по Риму-  
ману

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 все 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'e` или `Underfull'e`.

В следующем примере блоки, созданные командой `\parbox`, для наглядности взяты в рамку (с помощью `\fbox`):

Ку-ку.	Ку-ку.	<pre> \fbbox{% \parbox[b][2.5\height]{2cm}{% Ку-ку.}}\quad \fbbox{% \parbox[t][2.5\height]{2cm}{% Ку-ку.}}</pre>
--------	--------	--

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 в его аргументе допустимы пустые строки и команды `\par`, а 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. VI.4).

Коль скоро каждый блок, создаваемый ЛАТ<sub>Э</sub>X'овскими командами, рассматривается Т<sub>Э</sub>X'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:

<p>Внутри Т<sub>Э</sub>X'овских блоков может присутствовать не только собственно текст или формулы, но и другие блоки, и так далее.</p>
---

Просто поместить этот текст в аргумент команды `\fbox` не получится, поскольку наш текст в одну строку не укладывается, а команда `\fbox`,

подобно команде  $\text{\mbox}$ , текстов, не укладывающихся в строку, не переваривает. Поэтому нужно сделать из нашего абзаца блок с помощью команды  $\text{\parbox}$  и этот блок (т.е. уже «букву») передать в качестве аргумента команде  $\text{\fbox}$ :

```
\fbox{%
\parbox{6cm}{%
Внутри  $\text{\TeX}$ '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  $\text{\TeX}$  решил бы, что аргумент команды  $\text{\fbox}$  имеет пробел до и после «буквы», созданной командой  $\text{\parbox}$ . См. с. 12 по поводу использования знака процента для удаления нежелательных пробелов.

## 2.4. Сдвиги относительно базисной линии

Когда при исполнении команды  $\text{\makebox}$  или  $\text{\mbox}$   $\text{\TeX}$  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  $\text{\mbox}$ ), 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  $\text{\LaTeX}$ 'овской командой  $\text{\raisebox}$ . Эта команда требует двух обязательных аргументов. Первый из них — расстояние, на которое сдвигается по вертикали фрагмент текста, второй — сам этот фрагмент текста. Пример:

Слово подскочило в строке.      Слово  $\text{\raisebox{2pt}}{\text{подскочило}}$   
в строке.

Текст, расположенный во втором обязательном аргументе этой команды, должен удовлетворять тем же требованиям, что и аргумент команды  $\text{\mbox}$ : в нем могут быть самые разные  $\text{\TeX}$ 'овские команды, при условии, что среди них не будет команд типа пустой строки,  $\text{\par}$ ,  $\text{\ll}$  и тому подобных, которые «не укладываются в строку» (зато в этом тексте, как водится, могут присутствовать любые команды, порождающие блоки, в частности, например,  $\text{\parbox}$ , а уж в ее аргументе можно оставлять сколько угодно пустых строк). Если первый обязательный аргумент команды  $\text{\raisebox}$  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. Ы  
Вторая

Третья строка.

Строка. \\

Вторая

`\raisebox{7pt}[1pt][10pt]{Ы}\\`

Третья строка.

Буква Ы, поднятая на 7 пунктов над строкой, наложилась на первую строку, так как 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'a) высоту и/или глубину блока, не сдвигая его относительно базисной линии. В гл. VIII мы увидим пример такого использования этой команды.

### 3. Команда `\hbox`

В этом и следующем разделах мы рассмотрим средства генерации блоков, предоставляемые непосредственно языком TeX и макропакетом Plain TeX. Мы расскажем далеко не все (книгу [2] ничто не заменит), но сообщим тот минимум сведений, который необходим для модификации L<sup>A</sup>TeX'овского стандартного оформления, о чем пойдет речь в следующей главе. Всеми описываемыми в этом и следующем разделах TeX'овскими средствами можно пользоваться в L<sup>A</sup>TeX'овских исходных текстах.

Прежде всего вспомним (с. 125), 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 L<sup>A</sup>TeX'овской команды для генерации блока или линейки (к таковым относятся `\mbox`, `\makebox`, `\fbox`, `\framebox`, окружения `array`, `tabular` или `picture`, а также команда `\rule`<sup>1)</sup>) TeX из вертикального режима выходит и начинает очередной абзац.

Одна из основных TeX'овских команд для генерации блоков называется `\hbox`. В своем простейшем виде она полностью аналогична L<sup>A</sup>TeX'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.

---

<sup>1)</sup>Но не `\hrule` или `\vrule`: это команды TeX'a, а не L<sup>A</sup>TeX'a.



Сравните с тем, что было бы при использовании L<sup>A</sup>T<sub>E</sub>X'овской команды `\mbox` вместо `\hbox`:

На странице уже присутствует абзац текста. После того, как он кончится, T<sub>E</sub>X 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 T<sub>E</sub>X'овским `\hbox` и L<sup>A</sup>T<sub>E</sub>X'овским `\mbox`. Теперь поговорим о дополнительных возможностях, предоставляемых T<sub>E</sub>X'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 T<sub>E</sub>X'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 — T<sub>E</sub>X их проигнорирует<sup>2</sup>. Наконец, отсутствие `backslash` в слове `to` не является опечаткой: это не команда, а одно из «ключевых слов» T<sub>E</sub>X'a (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

---

<sup>2</sup>Пустых строк, однако, быть не должно.

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	<code>\hbox {Два слова}</code>
Два слова	<code>\hbox {Два \hfil слова}</code>
Два слова	<code>\hbox to 2cm {Два \hfil слова}</code>
Два слова	<code>\hbox to 3cm {Два \hfil слова}</code>
Два слова	<code>\hbox to 4cm {Два \hfil слова}</code>

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 несколько `\hfil`'ов, то растяжение произойдет на месте каждого из них, 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`'а, то на месте каждого из них будет добавлен пробел в 2,5 см. Вот пример с несколькими `\hfil`'ами:

Раз	два	три	<code>\hbox to 4cm{Раз \hfil два \hfil три}</code>
-----	-----	-----	--

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	<code>\hbox to 0.7\textwidth</code>
	<code>{Слева\hfil}</code>
Справа	<code>\hbox to 0.7\textwidth</code>
	<code>{\hfil Справа}</code>
В центре	<code>\hbox to 0.7\textwidth</code>
	<code>{\hfil В центре\hfil}</code>

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	<code>\hbox to 4cm{\hfil Слово\hfil}</code>
Слово	<code>\hbox to 4cm{\hfill Слово\hfil}</code>
Слово	<code>\hbox to 4cm{\hfil Слово\hfill}</code>

### 3.2. Отточия

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 L<sup>A</sup>T<sub>E</sub>X'овской команда `\dotfill`. Она работает так же, как и `\hfill`, с той разницей, что пробел, образующийся в результате действия этой команды, заполняется точками:

A.....B	<code>\hbox to 3cm{A\dotfill B}</code>
---------	--

Кроме этого, есть L<sup>A</sup>T<sub>E</sub>X'овская команда `\hrulefill`, которая также действует аналогично команде `\hfill` и при этом заполняет пробел линейкой:

1_____2_____3	<code>\hbox to 5cm{1\hrulefill 2\hrulefill 3}</code>
---------------	--

В T<sub>E</sub>Xнической 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 ЪЪЪЪЪЪЪЪЪЪЪЪЪЪЪЪ2	<code>\hbox to 5cm{1\leaders \hbox{Ъ}\hfil 2}</code>
---------------------	--

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 Т<sub>Е</sub>X’овская команда для генерации блока, например, `\hbox`, с которой мы уже познакомились, или `\vbox` или `\copy`, о которых еще пойдет речь. Команды Л<sup>A</sup>T<sub>Е</sub>X’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 Л<sup>A</sup>T<sub>Е</sub>X’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 Т<sub>Э</sub>Х поведет себя в столь странной ситуации)<sup>3</sup>. Здесь **plus** и **minus** — это, как мы помним, очередные ключевые слова Т<sub>Э</sub>Х’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 Т<sub>Э</sub>Х создать блок, ширина которого отличается от естественной, то для достижения требуемой ширины размеры пробелов будут изменяться. В Т<sub>Э</sub>Х’ической 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 Т<sub>Э</sub>Х 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 естественной, то Т<sub>Э</sub>Х вычисляет, насколько больше, после чего распределяет эту добавку между всеми пробелами пропорционально величинам **plus**-компонент клея в этих пробелах.

Пусть, скажем, мы создаем блок с помощью команды

```
\hbox to a {\hspace{0pt plus 2em}%
B\hspace{1cm plus 1em minus 2mm}B}
```

где величина  $a$  на 13 мм больше суммы ширин букв А, Б и В. Тогда 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,

---

<sup>3</sup>Если мы заставим такую пружину растянуться или сжаться больше, чем сказано, то получим сообщение «`Underfull \hbox`» или «`Overfull \hbox`»; см. ниже.

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. 111, 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.

Есть и более тонкие, чем `\hfil` или `\hfill`, способы задать бесконечно растяжимый клей. Именно, если сказать `\hspace{0pt plus 1fil}`, то в строку вставится клей с нулевой естественной шириной и бесконечной растяжимостью, а вот клей `0pt plus 3fil` имеет растяжимость хоть и тоже бесконечную, но в три раза большую, чем `0pt plus 1fil`, так что на него будет отведено в три раза больше места:

Слово
-------

```
\fbox{\hbox to 4cm
{\hspace{0pt plus 1fil}Слово%
\hspace{0pt plus 3fil}}}
```

Коэффициент перед `fil` может быть любой десятичной дробью. Можно также вместо `fil` с коэффициентом использовать `fill` с произвольным коэффициентом. Растяжимость у `fill` «еще более бесконечна», чем у `fil`, так что при совместном использовании клея с `fil` и клея

с `fill` все `fil`-компоненты будут проигнорированы (как в примере на с. 258). Наконец, отметим, что естественную ширину клея, использующего `fil` или `fill`, не обязательно делать нулевой: записи наподобие `\hspace{2cm plus 3fil}` также вполне законны.

### 3.4. Бесконечно сжимаемые интервалы

Мы уже два раза упомянули про клей с бесконечной сжимаемостью. Из многих способов его создавать 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	<code>\hbox to 50pt {Кот\hss Пес}</code>
КотПес	<code>\hbox{Кот\hss Пес}</code>
КотПес	<code>\hbox to 30pt {Кот\hss Пес}</code>
Кот	<code>\hbox to 15pt {Кот\hss Пес}</code>
ПесКот	<code>\hbox to 0pt {Кот\hss Пес}</code>

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 требуемому 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? 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. Существует даже Т<sub>Е</sub>X’овская команда `\llap`, которую можно определить так:

```
\newcommand{\llap}[1]{\hbox to 0pt{\hss #1}}
```

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, Т<sub>Е</sub>X будет просто считать, что ширина блока равняется нулю — мы обманули Т<sub>Е</sub>X, убедив его, что наш текст не занимает места по горизонтали! Для такого обмана (к нему приходится прибегать нередко) предусмотрена специальная Т<sub>Е</sub>X’овская команда `\rlap`, определяемая так:

```
\newcommand{\rlap}[1]{\hbox to 0pt{#1\hss}}
```

### 3.5. Еще раз о линейках

В аргументе команды `\hbox` может присутствовать и Т<sub>Е</sub>X’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  $\text{\LaTeX}$  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 `0.5em`, 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. 12).

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}

```

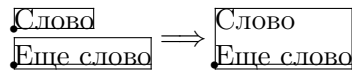
Мы воспользовались командой `\settowidth`, чтобы найти размер, который занимает линейка вместе с пробелом. Кстати, если просто написать `\hbox{\vrule\hspace{.5em}}`, то на печати мы ничего не увидим (внутри `\hbox`'а никакого текста нет, так что высота и глубина линейки равна нулю и она тем самым невидима); однако же эта команда создаст пробел, величина которого равна `0.4pt` плюс `0.5em`. Заключительное замечание: поскольку `flushleft`, 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	<code>\vbox{\hbox{Слово}}</code>
Еще слово	<code>\hbox{Еще слово}}</code>

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, что 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 подробности; для тех приложений,

которые мы имеем в виду, достаточно использовать команду `\vbox` так, как было предписано выше.

Когда  $\text{\TeX}$  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:

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. Блочные переменные

Наряду с переменными со значением длины, в  $\text{\LaTeX}$ е есть возможность определять переменные, значением которых является готовый  $\text{\TeX}$ овский блок, а затем эти переменные использовать.

Блочная переменная задается с помощью команды `\newsavebox`. Единственный аргумент этой команды — имя новой блочной переменной, которое должно удовлетворять тем же условиям, что любые имена

Однажды Лебедь, Рак и Щука...	<code>\sbox{\blok}{Рак}</code>
Однажды Лебедь,	<code>\copy\blok{}</code>
	и Щука\ldots

Обратите внимание, что при использовании команды `\copy` имя блочковой 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 `\hbox` или `\mbox`), эти две команды действуют совершенно одинаково, а вот будучи употребленным между абзацами, ЛАТЭХ'овское `\usebox` начинает новый абзац, в то время как ТЭХ'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 блочковую переменную, а затем в команде `\leaders` писать просто `\copy`. Пример:

```
*      *      *      *      *      *      \savebox{\blok}[1cm]{${*}$}
                                           \hbox to \textwidth
                                           {\leaders\copy\blok\hfil}
```

В этой ситуации по ТЭХ'ническим причинам сказать `\usebox` нельзя.

Скажем напоследок еще об одной конструкции, связанной с блочковыми переменными. Именно, если `\blok` — блочковая переменная, то можно «измерить» ширину, высоту и глубину блока, записанного в этой переменной, с помощью ТЭХ'овских команд `\wd`, `\ht` и `\dp`. Точнее говоря, сочетания `\wd\blok`, `\ht\blok` и `\dp\blok` можно использовать в точно-сти так же, как ТЭХ'овские параметры со значением длины, значения которых равны ширине, высоте и глубине блока:

```
12345                                           \sbox{\blok}{12345}\copy\blok
   345                                           \hbox to \wd\blok{\hfil 345}
   45                                           \hbox to \wd\blok{\hfil 45}
```

Для большинства простых приложений в ЛАТЭХ'е вполне хватает возможностей измерения блоков, предоставляемых командой `\settowidth` и ее аналогами, но иногда встречаются ситуации, в которых `\wd` удобнее.

## Глава VIII

# Модификация стандартных классов

Я думаю, гиппопотама  
Зовут так трудно 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 ЛАТ<sub>Е</sub>X’а. Возможно, вам даже захочется создать свой собственный класс документов вместо стандартных `article`, `proc`, `report` или `book`. Задача эта выполнимая, но для этого надо хорошо знать книгу [2] и исходные тексты ЛАТ<sub>Е</sub>X’а (они официально доступны). У читателя настоящей книги таких познаний не предполагается, так что мы предлагаем нечто более скромное: научиться модифицировать оформление одного конкретного документа.

Стандартное ЛАТ<sub>Е</sub>X’овское оформление не всех устраивает, и при этом в ЛАТ<sub>Е</sub>X’овском стандарте почти не предусмотрено удобных способов это оформление модифицировать. В прежние годы приходилось делать так: выяснить, как определяется, скажем, команда `\section`, понять, что в ее определении надо изменить для получения требуемого результата, и написать соответствующие макросы. В предыдущих изданиях этой книги автор полностью следовал такому подходу: все рекомендации, содержащиеся в этой главе, были найдены в результате изучения ЛАТ<sub>Е</sub>X’овских исходных текстов. К десятым годам XXI века ситуация серьезно изменилась, причем сразу в двух отношениях. С одной стороны, Т<sub>Е</sub>Xники со всего мира написали большое количество стилевых пакетов, предоставляющих массу возможностей для модификации оформления. С другой стороны, с развитием интернета и поисковиков эти пакеты стали лег-

кодоступны. В случае с  $\text{\LaTeX}$ 'ом, впрочем, нет даже большой нужды прибегать к услугам поисковиков: все полезные и не очень стилевые пакеты собраны на сайте CTAN (Comprehensive  $\text{\TeX}$  Archive Network) (<http://www.ctan.org>; если адрес вдруг изменится, то придется воспользоваться вышеупомянутыми поисковиками). На этом сайте имеется внутренний поиск, с помощью которого по ключевым словам можно найти ссылки на, возможно, подходящие вам стилевые пакеты. Чтобы выяснить, действительно ли такой-то пакет подходит, достаточно всего лишь почитать его документацию, имеющуюся по ссылке. С высокой вероятностью то, что нужно, найдется, причем в нескольких вариантах. Дополнительно устанавливать сам пакет, скорей всего, не придется: если вы установили себе  $\text{\TeX}$  в максимальной комплектации, то, наверное, он у вас уже есть. (Если вам понадобилось устанавливать пакет, отсутствующий на компьютере, прочтите в приложении О, как это делается.) В разд. 3.3 мы приводим пример поиска и использования готового пакета с CTAN'a.

Конечно, эта идиллическая картинка не обойдется без ложки дегтя: в стилевом пакете, который вы выбрали, может обнаружиться ошибка, а кроме того, несколько стилевых пакетов могут начать конфликтовать друг с другом. В этом случае остается либо смириться, отказавшись от желаемой модификации оформления, либо попробовать другой стилиевой пакет, обещающий добиться нужного вам эффекта, либо, если есть время и желание, разобраться, откуда берется ошибка, и исправить ее своими силами (для чего, видимо, понадобятся более серьезные познания в  $\text{\TeX}$ 'е).

## 1. С чего начать

Начнем с двух предупреждений. В этой главе мы расскажем вам, как можно довольно сильно изменить стандартное  $\text{\LaTeX}$ '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  $\text{\LaTeX}$ 'а. Ни в коем случае не меняйте ничего в этих файлах: все из-

менения в стиле оформления надо записывать в собственном стилевом файле, как объяснено ниже.

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 придется иметь дело со специальными командами  $\text{\LaTeX}$ 'а, содержащими в своем имени символ @. Поскольку @ — не буква, просто так до этих команд не добраться<sup>1</sup>. 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{mystyle}` 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 эти пакеты подключить. При этом нужно использовать команду `\RequirePackage` (вместо знакомой вам `\usepackage`), например, так:

```
\RequirePackage{array,longtable}
\RequirePackage[intlimits]{amsmath}
```

(необязательный аргумент команды `\RequirePackage` означает то же самое и используется так же, как у команды `\usepackage`).

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 статус-кво. Если вы использовали в тексте `\makeatletter`, не забудьте написать и `\makeatother` сразу после текста, в котором использовалась @ в имени команд.

---

<sup>1</sup>Если просто написать `\@addtoreset` (скоро вы узнаете, что это значит), то  $\text{\TeX}$  воспримет это как команду `\@` (имя — из одной не-буквы!), за которой следует текст `addtoreset`.



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. 230 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.

Иногда бывает нужно, наоборот, вывести один счетчик из подчинения другому. В стандартном L<sup>A</sup>T<sub>E</sub>X'e такой команды (даже с @ в имени) не предусмотрено, но она определена в стилевом пакете `remreset`. Именно, если этот пакет подключен, то, например, в классе `book` команда

```
\@removefromreset{footnote}{chapter}
```

приведет к тому, что нумерация сносок будет сплошной по всему тексту (а не будет начинаться заново в каждой главе). Так как команда `\@removefromreset` содержит в своем имени символ @, пользоваться ей можно только в стилевом файле (или после `\makeatletter`).

При ознакомлении с командой `\@addtoreset` может возникнуть искушение написать

```
\@addtoreset{footnote}{page}
```

чтобы сноски нумеровались заново на каждой странице. К сожалению, по Т<sub>Е</sub>Хническим 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.

Разумеется, для того чтобы осмысленно применять описанные приемы, надо знать, какие счетчики определены в стандартных стилях и кому они подчинены (или не подчинены). Эта информация содержится в конце раздела.

## 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. VI 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`. 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. Кто кому подчинен в стандарте

Нам осталось выполнить свое обещание и рассказать, какие счетчики определены в L<sup>A</sup>T<sub>E</sub>X'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 сносков 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. VI.2.5, 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 L<sup>A</sup>T<sub>E</sub>X'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 L<sup>A</sup>T<sub>E</sub>X'овская команда для раздела документа дана без звездочки — иначе нумерации не будет заведомо). Уровни вложенности в стандартных стилях приведены в табл. VIII.1. Стало быть, если мы хотим, чтобы

Таблица VIII.1. Уровни вложенности разделов документа

Название раздела	Уровень
<code>\section</code>	1
<code>\subsection</code>	2
<code>\subsubsection</code>	3
<code>\paragraph</code>	4
<code>\subparagraph</code>	5

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 ЛАТЭХ'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<sup>2</sup>, 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 ЛАТЭХ 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

<sup>2</sup>У читателя может возникнуть недоумение: а откуда же команда `\section` узнает название раздела? Ответ: команда `\@startsection` на самом деле принимает не шесть, а семь аргументов, седьмой из которых — как раз название: если написать `\section{0 слонах}`, то после разворачивания макроса `\section` возникнет команда `\@startsection` с шестью аргументами, указанными выше, и седьмым — заглавием. Впрочем, и это не вся правда: надо еще предусмотреть обработку возможного необязательного аргумента команды `\section`, о чем мы почли за благо умолчать.

(см. разд. 6). Ясно, что без особой 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 не что иное, как `-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 ЛАТЭХ'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. 130).

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 уместается). 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;

Таблица VIII.2. Стандартное оформление разделов

	Отступ	Перед	После
<code>\subsection</code>	Opt	-3.25ex plus -1ex minus-.2ex	1.5ex plus .2ex
<code>\subsubsection</code>	Opt	-3.25ex plus -1ex minus-.2ex	1.5ex plus .2ex
<code>\paragraph</code>	Opt	3.25ex plus 1ex minus.2ex	-1em
<code>\subparagraph</code>	<code>\parindent</code>	3.25ex plus 1ex minus.2ex	-1em

- 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 ЛАТ<sub>E</sub>X'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 ЛАТ<sub>E</sub>X'овских параметров оформления, наподобие отступа перед или после заголовком. Для команды `\section` мы привели их выше, а для остальных стандартных команд рубрикации они собраны в табл. VIII.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. VIII.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 или модифицирует его самостоятельно.

Шестой аргумент команды `\@startsection` позволяет автоматически добавлять текст перед номером раздела. К сожалению, не так просто уговорить  $\text{\LaTeX}$  автоматически добавлять текст после номера, хотя такая потребность порой возникает (например, хочется, чтобы в заголовках разделов после номеров стояли точки, чего стандартное  $\text{\LaTeX}$ ’овское оформление не предусматривает). В разд. 3.3 ниже мы расскажем, как для этих целей воспользоваться готовым стилевым пакетом.

Если заставить  $\text{\LaTeX}$  ставить точки после номеров, написав

```
\renewcommand{\thesection}{\arabic{section}.}
```

то 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  $\text{\LaTeX}$ ’овских классах иначе. Не будем вдаваться в подробности, как именно, а вместо этого рассмотрим единственно важный для нас вопрос: как можно менять оформление глав.

Большая 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  $\text{\TeX}$ ’а на  $\text{\LaTeX}$  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. 154). 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. Если вы захотите в этой команде что-то изменить, то скорее всего это будет 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!

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.

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 ЛАТЭХ с ТЭХ'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 ТЭХ'овских и ЛАТЭХ'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.

### 3.3. Использование готовых пакетов

Мы много чего научились делать с заголовками, но так и не смогли поставить точку после их номеров. Давайте посмотрим, как найти стилевой пакет, который сделает эту работу за нас. Прошу поверить, что все описываемые ниже действия я проделал непосредственно перед написанием этого текста.

Итак, заходим на CTAN (<http://www.ctan.org>); поскольку нас интересуют заголовки разделов, задаваемых командой `\section`, в окошке поиска набираем «section headings». Мне на этот запрос выпало аж 19 ссылок. Не мудрствуя лукаво, пойдем по ним подряд.

Самая первая ссылка «chapter, section, etc., heading styles» ведет не к конкретному стилевому пакету, а к другому списку ссылок. По-временам с ней. Вторая ссылка — «section» — ведет уже на страницу конкретного стилевого пакета `section`. На этой странице есть ссылка на документацию к пакету. Это pdf-файл длиной 9 страниц; беглый взгляд на него показывает, что возможностей у стилевого пакета довольно много, но и разбираться в документации придется довольно долго: по крайней мере, ни одной четкой инструкции сходу не заметно. Посмотрим, что есть еще. Следующая ссылка ведет на страницу пакета `varsects`, на которой почему-то нет ссылки на документацию. А вот на четвертой ссылке нам, похоже, повезло: она ведет на страницу пакета под названием `secdot`. Перейдя с этой страницы по ссылке на pdf-файл с документацией, состоящий всего из одной страницы, мы обнаруживаем, что это именно то, что нам нужно: в первой же строке там сообщается, что при подключении этого пакета после номеров разделов, определенных как `\section`, будут печататься точки. Эксперимент с коротеньким пробным файлом показывает, что так и есть. Впрочем, успокаиваться рано: если в пробном файле создать еще и `\subsection`, то после ее номера точка так и не появляется. Ничего страшного: если прочесть документацию чуть дальше, то выясняется, что, подключив этот пакет, можно сказать `\secdot{subsection}`, в результате чего точки после номеров появятся и у разделов типа `\subsection`. Более того, согласно документации, пакет предоставляет возможности и для более гибкой настройки: в пакете определяется еще команда `\sectionpunct` с двумя аргументами, первый из которых — `section`, `subsection` и т. п., а второй — то, что будет поставлено после номера раздела (согласно документации, по умолчанию это `.\quad` — и можно, например, попробовать изменить горизонтальную отбивку).

Вот, собственно, и все, что умеет этот стилевой пакет<sup>3</sup>. Ни на шрифт, которым печатаются заголовки, ни на отбивки вокруг этих заголовков,

---

<sup>3</sup>На самом деле в документации еще объясняется (способом, малопонятным для

ни на абзацный отступ в первом абзаце с помощью пакета `secdot` повлиять невозможно.

А что же делать, если хочется не только точку после номера поставить но и еще как-то изменить оформление заголовков или отбивки вокруг них? В данном случае нам повезло: пакет `secdot` действительно можно использовать совместно с рецептами из предыдущего пункта (или, коль на то пошло, с пакетом `section`, в документации к которому мы сходу не разобрались). Но вообще говоря, так будет не всегда: вполне может оказаться, что два стилевых пакета, прекрасно работающие по отдельности, при совместном подключении друг другу мешают. Выяснить, так ли это в каждом конкретном случае, можно только экспериментально. Общий совет может быть только один: при модификации оформления с помощью готовых стилевых пакетов, как и при модификации вручную, лучше соблюдать умеренность.

Тем, кто умеренность в данном вопросе проявлять не намерен, но, напротив, планирует радикально изменить надоевшее стандартное оформление заголовков к разделам, можем порекомендовать обладающий весьма широкими возможностями стилевой пакет `titlesec`.

## 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  $\text{\LaTeX}$ '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.

Начнем со следующего замечания. Если русский текст оформляется с использованием пакета `inputenc`, то в `toc`-, `lot`- и `lof`-файлах вы увидите не русские буквы, а набор загадочных  $\text{\TeX}$ 'овских команд, что работу отнюдь не облегчает. Уж если вы занялись модификацией стандартного оформления, то настоятельно рекомендуем оформлять русские тексты без использования `inputenc` — так, как рассказано в приложении И.

Теперь перейдем собственно к рассказу о работе над оглавлением. Для начала объясним, как составлять оглавление полностью вручную,

---

начинающих  $\text{\TeX}$ ников), как с помощью пакета `secdot` можно задать произвольное оформление номера раздела. Но вот это уже действительно все.

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. Вторым аргументом — то, что вы хотите записать в файл. Если, например, вам взбрело в голову внести в оглавление к книге текст «У попа была собака», 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. 155). 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` (самой 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  $\text{\LaTeX}$ '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}{0 слонах} (*)
```

и если эта команда попала на страницу 95, то при следующем запуске  $\text{\LaTeX}$ 'а в процессе чтения `toc`-файла будет исполняться команда

```
\l@abcd{0 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. 0 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  $\text{\LaTeX}$  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  $\text{\TeX}$  отведет на номер раздела. Результат выйдет 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@section` и «более мелкие» определяются в классе `book` так же, как `\l@section`, отличаются только аргументы команды `\@dottedtocline` — см. табл. VIII.3.

Таблица VIII.3. Стандартные определения `\l@`-команд (класс `book`)

	Три аргумента <code>\@dottedtocline</code>		
<code>\l@subsection</code>	2	3.8em	3.2em
<code>\l@subsubsection</code>	3	7.0em	4.1em
<code>\l@paragraph</code>	4	10em	5em
<code>\l@subparagraph</code>	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; 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. 113) 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

$$\text{\leftskip} + \text{\parfillskip} = \text{\@pnumwidth} - \text{\@pnumwidth} = 0,$$

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 ЛАТЭХ'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 ЛАТЭХ'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<sup>4</sup> такой текст (будем считать, что глава называется «Все о слонах»):

```
\makebox[\@tempdima][l]{\thechapter}Все о слонах
```

---

<sup>4</sup>Начиная с левого поля, невзирая на установленное значение `\leftskip`; именно для этих целей в определение включена команда `\hspace{-\leftskip}`.

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  $\text{\LaTeX}$  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}}
```

Здесь `\contentsname` — это уже знакомый нам макрос, разворачивающийся в заголовок оглавления. Как видите, заголовок оглавления оформляется просто как заголовок нумерованного 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 колонтитутлы (вспомним, что `\section*` сама по себе никакой информации для колонтитутлов 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

Сейчас мы завершим рассказ о том, как менять стиль оформления перечней (см. разд. VI.2.5). Не с легким сердцем приступает автор к изложению этой темы.

По мнению многих пользователей, установленное по умолчанию оформление окружений `itemize`, `enumerate` и `description` довольно неудачно (особенно раздражают излишне большие вертикальные отбивки между элементами перечня), так что совершенно естественно желание это оформление изменить. К несчастью, при этом никаких удобных средств для изменения данных элементов оформления в  $\text{\LaTeX}$ 'е не предусмотрено: приходится выкручиваться, прибегать к «грязным трюкам», и даже после этого желаемый эффект достигается не во всех случаях. Если вы все же решились читать дальше, запаситесь терпением, а в процессе чтения не делайте скоропалительных выводов.

В гл. III мы называли перечнями окружения `itemize`, `enumerate` и `description`; помимо этого, к перечням в  $\text{\LaTeX}$ 'овском смысле относятся `flushleft`, `flushright`, `center`, `quote`, `quotation`, `verse`, `tabbing`, а также окружения, создаваемые с помощью команды `\newtheorem`; как мы увидим в разд. 5.3, все эти окружения — частные случаи одной  $\text{\LaTeX}$ '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. В двух следующих подразделах описаны несколько способов добиться того, чтобы такие изменения подействовали. У каждого из этих способов есть существенные недостатки.

Теперь договоримся о терминологии. Каждый перечень ЛАТ<sub>E</sub>X 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. 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 ЛАТ<sub>E</sub>X 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

$$\text{\labelmargin} - \text{\labelwidth} - \text{\labelsep} \quad (*)$$

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, аб-

зацах 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. 130), 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. 132).

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 можно попытаться поменять вышеописанные параметры. Для всех этих параметров имеются значения по умолчанию, устанавливающиеся в момент исполнения команды `\documentclass`, так что хочется просто присвоить им желаемые значения в своем стилевом файле или в преамбуле документа. С некоторыми из этих параметров такой прием сработает, но с некоторыми другими — нет. В частности, не сработает это и с параметром `\itemsep`, который в первую очередь хочется изменить. Что же делать? Ниже мы предлагаем несколько решений, каждое из которых имеет свои недостатки.

Во-первых, заведомо сработает следующий примитивный прием: сразу же после `\begin{itemize}` записывать команды для установки новых значений параметров списка (то же относится и к окружениям `enumerate`, `description`, и вообще к любым окружениям, являющимся L<sup>A</sup>T<sub>E</sub>X'овскими списками — не будем это всякий раз оговаривать).

Разумеется, вместо того чтобы всякий раз писать, например,

```
\begin{itemize}
\itemsep=0pt plus 1pt
\item ...
```

можно применить «малую механизацию»: определить новое окружение (назовем его, скажем, `myitemize`) как

```
\newenvironment{myitemize}{%
\begin{itemize}
\setlength{\itemsep}{0pt plus 1pt}}{%
\end{itemize}}
```

после чего всюду писать `{myitemize}` вместо `{itemize}`

Этим бы, честно говоря, можно было ограничиться, но для любознательного читателя расскажем кое-что еще. У описанного выше приема модификации перечней два очевидных недостатка: во-первых, для каждого типа перечней приходится определять отдельное новое окружение, и во-вторых, пользоваться стандартными именами для списков, вроде `enumerate`, при этом нельзя. Вот бы написать в стилевом файле такие волшебные слова, чтобы не пришлось ничего переименовывать, а при желании еще раз изменить оформление перечней можно было сделать это одним махом для всех их типов! Ну что ж, давайте попробуем

Для начала надо понять, почему может не сработать присваивание значений параметрам перечня в преамбуле или стилевом файле. Механизм тут следующий. При каждом «входе» в перечень ЛАТ<sub>Э</sub>X в первую очередь выясняет его уровень вложенности: 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. При этом единственное действие, которое выполняют команды `\@listi`, `\@listii` и т. п. — присваивание значений некоторым параметрам перечня, перечисленным в предыдущем пункте.

Вот, например, как определяется (опять «в переводе с Т<sub>Э</sub>X’а на ЛАТ<sub>Э</sub>X») команда `\@listi` в классе `article` с опцией `11pt`<sup>5</sup>:

```
\newcommand{\@listi}{%
\leftmargin=25pt
\topsep=8pt plus 2pt minus 4pt
\itemsep=4pt plus 2pt minus 1pt }
```

---

<sup>5</sup>В случае, когда размер текущего шрифта «нормальный» (как после исполнения команды `\normalsize`). См. ниже.



Прочих параметров `\@listi` не трогает, а значения `\leftmargin`, `\topsep` и `\itemsep`, устанавливаемые согласно этому определению, совпадают со значениями по умолчанию. Теперь ясно, что если просто написать `\itemsep=0pt` в преамбуле или стилевом файле, то никакого действия это не возымеет.

Пусть теперь мы хотим, скажем, в перечнях первого уровня установить `\itemsep` в ноль пунктов, а остальные параметры трогать не собираемся. Первое, что приходит в голову — переопределить в стилевом файле команду `\@listi`, чтобы в ней было написано `\itemsep=0pt`. Увы, это тоже не сработает! Для достижения искомого эффекта надо в стилевом файле написать буквально следующее:

```
\renewcommand{\@listI}{}
\itemsep=0pt
```

(`\@listI` вместо `\@listi` — не опечатка). Теперь для списков первого уровня все<sup>6</sup> получится.

Читатель вправе спросить, что все это значит. Ответ таков. Во-первых, наряду с командой `\@listi` существует еще и команда `\@listI`, которая определяется *точно так же*, как `\@listi`. Во-вторых, при каждом исполнении команды `\normalsize` (устанавливающей, как мы помним, «обычный» размер шрифта) происходит, помимо прочего, следующее действие: команда `\@listi` переопределяется таким образом, что она становится по своему действию равносильной команде `\@listI`. Грубо говоря, можно считать, что в процессе исполнения команды `\normalsize` выполняется действие

```
\renewcommand{\@listi}{\@listI}
```

(на самом деле этот эффект достигается с помощью иных Т<sub>Е</sub>X'овских средств). Остается добавить, что команда `\normalsize` заведомо выполняется в начале обработки Л<sup>A</sup>T<sub>Е</sub>Xовского файла (она входит в комплекс действий, производимых при исполнении команды `\begin{document}`) — и станет ясно, что переопределение команды `\@listi` в стилевом файле ничего не дает (оно забудется сразу после `\begin{document}`), а вот переопределение `\@listI` — это ровно то, что нужно. Теперь при каждом входе в перечень первого уровня команда `\@listI` ничего менять (и вообще ничего делать) не будет, так что те установки, что мы сделали в преамбуле, останутся в силе.

Со списками, вложенными в другие списки (т.е. теми, у кого уровень вложенности больше единицы) все, можно сказать, просто и скучно: чтобы изменить параметры оформления списков второго, третьего

---

<sup>6</sup>С некоторыми исключениями, о которых речь пойдет ниже.

и т.д. уровня, надо переопределить в стилевом файле именно команду `\@listii`, `\@listiii` и т.д. Не забудьте, что в определениях этих команд могут присутствовать только присваивания значений параметрам списка.

Все сказанное выше относилось исключительно к спискам, печатаемым шрифтом нормального размера. Если, однако, вы попробуете, после всех указанных выше настроек, организовать список вроде `itemize` в области действия команды `\small`, то обнаружите, что ваши настройки не действуют. Дело в том, что не только команда `\normalsize` при своем выполнении переопределяет команду `\@listi`: так же поступают и некоторые другие команды смены размера шрифта (в стандартных ЛАТЭХ'овских классах — еще `\small` и `\footnotesize`). Собственно говоря, это вполне логично: если список набирается более мелким шрифтом, то и вертикальные отбивки в нем должны быть меньше. В отличие от `\normalsize`, другие команды смены размера шрифта переопределяют `\@listi` напрямую, а не через посредника (или вообще не трогают).

Все вышеописанное вряд ли можно назвать примером хорошего стиля программирования, но так уж ЛАТЭХ устроен, и менять его поздно.

Остается вопрос, что же все-таки делать с параметрами списков в присутствии команд `\small` или `\footnotesize`. Ответ: в рамках любой из двух предложенных методик остается только вручную устанавливать параметры после `\begin{itemize}` и подобных ей команд. Будем надеяться, что мелким шрифтом у вас набрана лишь малая часть текста.

Итак, мы описали два способа настройки параметров списка, и ни один из них не является вполне удовлетворительным. На самом деле существует полностью корректный способ настройки этих параметров, которым профессионалы обычно и пользуются: надо переопределить в своем стилевом файле команды для смены размера шрифта, заменив имеющиеся в них определения команды `\@listi` на нужные вам. Такое переопределение будет хорошим упражнением для желающих разобратся в предмете поглубже.

### 5.3. Окружения `list` и `trivlist`

Все ЛАТЭХ'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	<code>\begin{list}{И какой-то}{}</code>
показал ему шиш.	<code>\item малыш показал ему шиш.</code>
	<code>\item барбос укусил его в нос.</code>
И какой-то барбос	Нехороший барбос, невоспитанный!
укусил его в нос. Нехороший бар-	<code>\end{list}</code>
бос, невоспитанный!	

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, так что теперь мы узнали еще один — и опять неидеальный — способ изменить параметры перечня). 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:

А: Выглядит так же, как соответствующая русская буква, и читается так же.

В: Читается не так, как похожая на нее русская буква.

С: 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. 118).

Наряду с окружением `list` в ЛАТ<sub>Э</sub>X'e 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` часто добавляют команду `\item[]` (и тогда `\item` внутри окружения вообще не используют).

Окружения `list` можно вкладывать друг в друга; максимальная глубина вложенности равна шести. В окружениях наподобие `itemize` предусмотрен «ограничитель», снижающий эту максимальную 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 ЛАТ<sub>Э</sub>X'a, можно рассматривать как состоящую из трех частей: тела страницы, верхнего ко-

лонтитула и нижнего колонтитула<sup>7</sup>. Мы знаем, что с помощью команды `\pagestyle` на оформление колонтитулов можно в какой-то мере влиять. Возможностей для этого, однако же, не слишком много. В этом разделе мы расскажем, как можно радикально менять вид колонтитулов. Как водится, наряду с излагаемыми ниже рецептами существует и готовое решение — пакет `fancyhdr`, дающий возможность работать с колонтитулами, не вникая в «кухню», но предоставляющий меньше возможностей. Впрочем, для полноценной работы с этим пакетом также желательно понимание механизма передачи информации из текста в колонтитулы, описываемого ниже в разд. VIII.6.2.

### 6.1. «Статические» колонтитулы

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 `\@oddfirst` и `\@evenfirst`, отвечающими за нижние колонтитулы. Все четыре названные команды получают некоторое определение в L<sup>A</sup>T<sub>E</sub>X'овском стандарте, так что переопределять их надо с помощью `\renewcommand`.

Теперь обсудим, как вышеуказанные 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 T<sub>E</sub>X'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;

---

<sup>7</sup>Эта терминология не совпадает с традиционной, но удобна для наших целей.

- на правой странице разворота в верхнем колонтитуле написано название романа, прижатое (выключенное) влево;
- В нижних колонтитулах по центру расположен номер страницы, окруженный тире.

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{@oddfont}{\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{@oddfont}{}%
```

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.

## 6.2. Передача информации из текста в колонтитулы

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  $\text{\LaTeX}$ е используются команды `\markboth` и `\markright`. Разберем, как они работают. Команда `\markboth`, требующая двух обязательных аргументов, запоминает пару «пометок» — два фрагмента текста (возможно, с  $\text{\TeX}$ 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»<sup>8</sup>. 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:

---

<sup>8</sup>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.



`\markboth{u}{v}`, `\markboth{a}{b}` и `\markboth{m}{n}`,

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	<code>\leftmark</code>	<code>\rightmark</code>
1	<i>x</i>	<i>t</i>
2	<i>z</i>	<i>t</i>
3	<i>u</i>	<i>n</i>
4 и далее	<i>m</i>	<i>n</i>

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 L<sup>A</sup>T<sub>E</sub>X'овская команда `\MakeUppercase`, которую мы ранее не рассматривали. Не вдаваясь в подробности, скажем, что эта команда переводит все буквы в тексте<sup>9</sup>, 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. 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 раздела

---

<sup>9</sup>Но не в именах команд.

(наклонным шрифтом, прописными буквами) — выключенное вправо<sup>10</sup>: 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
{\MakeUppercase{\thesection\hspace{1em}#1}}% левая пометка
{\MakeUppercase{\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`.

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»:

---

<sup>10</sup>Точнее говоря, это будет заглавие либо текущего раздела, либо первого из разделов, начинающихся на этой странице — см. выше обсуждение `\leftmark` и `\rightmark`.

```
\renewcommand{\subsectionmark}[1]{}
```

Бывает и так, что вас устраивает стандартный стиль оформления колонтитулов, но при этом не устраивает, что 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  $\text{\LaTeX}$  предоставляет вам более простой способ. Если написать в стилевом файле (до того, как вы переопределяете команды наподобие `\@oddhead`)

```
\pagestyle{myheadings}
```

то все  $\text{\LaTeX}$ '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{0 некоторых специальных свойствах  
подмножеств пустых множеств, не рассматривавшихся  
в предыдущих разделах статьи}
```

Рассмотрим `\markboth{\thesection\hspace{1em}Подмножества}{}`  
пустое множество

(`\markboth` ставится без пробела после первого слова раздела!) — 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`). Если поставить `\markboth` сразу после команды `\section` или в ее аргументе, то возможны различные неприятности.

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 ЛАТ<sub>Э</sub>X'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`).

## 7. Плавающие объекты

Этот раздел посвящен плавающим иллюстрациям и таблицам (короче: плавающим объектам).

### 7.1. Оформление подписуночной 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 `\abovcaptionskip` (отбивка над подписью) и `\belowcaptionskip` (под подписью). Обе эти отбивки являются растяжимыми. 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{\abovcaptionskip}%
\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 Л<sup>A</sup>T<sub>E</sub>X'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 длиннее, чем ширина полосы). В следующих шести строках (с помощью не рассматривавшейся нами Т<sub>E</sub>X'овской конструкции «условных макросов») длина подписи с номером сравнивается с шириной текста (она обозначена Т<sub>E</sub>X'овским параметром `\hsize`: при одноколоном наборе это то же самое, что `\textwidth`, а при многоколоном — `\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 печатает-

ся 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 уместается, а текст между `\else` и `\fi` — к противоположному.

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.

Как и все прочее в L<sup>A</sup>T<sub>E</sub>X'e, оформление подрисовочных подписей можно менять не только вручную, но и с помощью готовых стилевых пакетов. Если вы предпочитаете пользоваться уже имеющимися разработками опытных T<sub>E</sub>Xников, к вашим услугам пакет `caption`, дающий широкие возможности для модификации такого оформления.

## 7.2. Размещение плавающих объектов на странице

Сейчас мы обсудим параметры, которыми руководствуется L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X'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 ЛАТЭХ'ом специально для размещения плавающих объектов при обработке необязательного аргумента **p**, это не относится). Значение по умолчанию: 0.2.

**\floatpagefraction** Этот параметр, напротив, относится к страницам, которые ЛАТЭХ 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 вверх или вниз 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 использовании 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 две 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 одна команда:

**`\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`<sup>11</sup>. Определять эти команды нужно не произвольным образом: чтобы они правильно стыковались с L<sup>A</sup>T<sub>E</sub>X'овскими алгоритмами размещения плавающих объектов, нужно иметь в виду следующее:

- 1) каждая из этих команда будет выполняться в те моменты, когда T<sub>E</sub>X находится в вертикальном режиме;
- 2) по окончании работы каждой из этих команд T<sub>E</sub>X должен снова оказаться в вертикальном режиме;
- 3) текст, генерируемый каждой из этих команд, не должен, с точки зрения T<sub>E</sub>X'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 L<sup>A</sup>T<sub>E</sub>X собьется со счета при решении вопроса о размещении иллюстраций. Формально правильным было бы такое решение:

<sup>11</sup>Именно так, а не переопределить с помощью `\renewcommand`!

```
\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 командами `\topfigrule` и `\dblfigrule` вы сможете теперь разобратся 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. 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 L<sup>A</sup>T<sub>E</sub>X'a 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 (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  $\TeX$ 'а, 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. 314.

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  $\TeX$ 'а, 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
$\sim\@thefnmark$}}
```

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.

## 8.2. 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 L<sup>A</sup>T<sub>E</sub>X'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 (не рассматривавшимся нами способом) материал для колон-титолов. 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 нумерованной 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. 133). 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).

### 8.3. Вертикальные отбивки вокруг выключных формул

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

<code>\abovedisplayskip</code>	<code>\abovedisplayshortskip</code>	(перед формулой)
<code>\belowdisplayskip</code>	<code>\belowdisplayshortskip</code>	(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).

К сожалению, изменить эти параметры так же непросто, как непросто изменить вертикальные отбивки между элементами перечней (см. разд. 5): они устанавливаются при выполнении командах для смены размера шрифта (`\normalsize`, `\small` и т. п.). Это логично (такие отбивки и должны зависеть от размера шрифта), но никаких готовых средств для модификации этих отбивок в  $\LaTeX$ 'е, к сожалению, нет. Так или иначе, в преамбуле менять параметры наподобие `\abovedisplayskip` бессмысленно: после первой же команды `\normalsize` (выполняемой, в частности, в процессе выполнения команды `\begin{document}`) все изменения теряются.

Единственно разумный способ модифицировать такие отбивки — переопределить команду `\normalsize` и ей подобные; это не чересчур сложно, но для этого требуется несколько большее знакомство с  $\TeX$ 'ом, чем у читателей нашей книги. Для желающих изучить  $\TeX$  поглубже такое переопределение будет посильным и полезным упражнением.



## Приложение А. Архитектура $\text{\TeX}$ 'а и $\text{\LaTeX}$ '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  $\text{\TeX}$ 'а. Непонятные места можно спокойно пропускать.

### А.1. Немного истории

Как гласит легенда, Дональд Кнут (Donald E. Knuth) написал программу  $\text{\TeX}$  для себя, чтобы готовить к печати книги серии «Искусство программирования для ЭВМ». Первая версия  $\text{\TeX}$ 'а появилась в 1978 году; начиная с 1989 года все изменения в (классическом)  $\text{\TeX}$ 'е сводятся к исправлению ошибок: Кнут сознательно не стремится совершенствовать  $\text{\TeX}$  дальше, чтобы предотвратить распространение несовместимых версий.

Тому, кто первым обнаруживал очередную ошибку, Кнут выплачивал денежную премию; эта премия время от времени удваивалась и выросла от \$ 2.56 ( $2^8$  центов) до \$ 327.68 ( $2^{15}$  центов)<sup>1</sup>, после чего Кнут начал новый отсчет. В любом случае ошибок в  $\text{\TeX}$ 'е за все эти годы обнаружено очень немного.

Начиная с версии 3.0, версии  $\text{\TeX}$ 'а нумеруются последовательными десятичными приближениями числа  $\pi$ ; по состоянию на январь 2014 года текущая версия  $\text{\TeX}$ 'а имела номер 3.14159265. Кнут просит после его смерти присвоить последней на тот момент версии номер  $\pi$ , после чего дальнейшее совершенствование  $\text{\TeX}$ 'а прекратить, переведя остающиеся ошибки (bugs) в разряд особенностей (features).

Программа  $\text{\TeX}$  распространяется свободно, а ее исходные тексты полностью доступны: помимо книги [2], обязательной для изучения каждому, кто собирается стать  $\text{\TeX}$ ником, в виде книги опубликован и полный текст программы с комментариями. Текст программы написан на разработанном Кнутом языке под названием Web, к которому Кнут создал два транслятора: один из текста, написанного на Web'е, получает программу на Pascal'е (скомпилировав которую, можно получить исполняемый файл программы  $\text{\TeX}$ ), а другой из того же самого Web-файла получает  $\text{\TeX}$ 'овский файл, представляющий собой текст программы с комментариями (обработав который  $\text{\TeX}$ 'ом, можно получить програм-

---

<sup>1</sup>Большинство получателей премиальных чеков оставляли их в качестве сувениров. Когда сканы чеков начали выкладывать в интернет, Кнут, опасаясь кибермошенников, стал высылать чеки на банк островного государства San Serriffe, изобретенного журналистами одной из британских газет к 1 апреля 1977 года; Д. Кнут обещает выплатить деньги тем, кто захочет получить премию именно деньгами.

му с комментариями, изданную в виде книги). В настоящее время исходные тексты на Web'е транслируют не в Pascal, а в язык C.

## А.2. Макропакеты и форматы

Говоря техническим языком,  $\TeX$  представляет собой интерпретируемый язык программирования. Это означает, что любой символ (или последовательность символов) в  $\TeX$ -файле воспринимается как команда: буква **z** означает «напечатать букву **z**», пустая строка или последовательность символов `\par` — «завершить абзац», и т. п.

Если, однако, получить программу  $\TeX$  непосредственно из кнотовского исходного текста (то есть перевести Web в Pascal или C, а затем скомпилировать), то использовать полученную программу («чистый  $\TeX$ » — по-английски «*virgin  $\TeX$* ») для обработки текстов будет еще невозможно: чистый  $\TeX$  не знает смысла ни спецзнаков (кроме «`\`»), ни макросов. В него встроены только «примитивные команды» (с. 219). Чтобы использовать  $\TeX$  для дела, надо предварительно задать назначение различных спецзнаков и — главное — определить большое количество макросов. Это также делается средствами чистого  $\TeX$ 'а: в число примитивов входят команды для определения новых макросов (наподобие известной вам `\newcommand`, но с иным синтаксисом и более широкими возможностями). Стало быть, наш файл придется начать с длинной цепочки определений макросов.

Разумеется, делать так каждый раз неразумно: проще, чтобы квалифицированный человек написал эти определения раз и навсегда, записал их в специальный файл, а мы затем, например, включили этот файл в начало своего текста с помощью команды `\input` (в чистом  $\TeX$ 'е это тоже примитив). Такой заготовленный заранее 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.

Впрочем, в реальности так никогда не делается: макропакеты имеют достаточно большой размер, и при компьютерных мощностях конца 70-х годов XX века потери времени при работе по такой схеме были бы неприемлемы. Поэтому Кнут применил 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, вместо того чтобы всякий раз обрабатывать исходный текст макропакета заново. Более того, таблицы переносов (необходимые для верстки

абзацев и также неизвестные «чистому»  $\text{\TeX}$ 'у) могут быть обработаны  $\text{\TeX}$ 'ом только на стадии генерации форматного файла (еще одна оптимизация). Так что, строго говоря,  $\text{\TeX}$  — это интерпретируемый язык с элементами компиляции.

Первый макропакет написал сам Кнут одновременно с программой  $\text{\TeX}$ : это пакет Plain  $\text{\TeX}$ , также описанный в книге [2].

Следующим после Plain  $\text{\TeX}$ 'а макропакетом, получившим распространение, стал созданный Майклом Спиваком (Michael Spivak)  $\mathcal{A}\mathcal{M}\mathcal{S}\text{\TeX}$ . Это набор макроопределений в дополнение к Plain  $\text{\TeX}$ 'у, включающий удобные средства для набора сложных формул, особенно многострочных. Подчеркнем, что всех  $\mathcal{A}\mathcal{M}\mathcal{S}\text{\TeX}$ 'овских эффектов можно в принципе добиться и в Plain  $\text{\TeX}$ 'е — возможности  $\text{\TeX}$ 'а новый макропакет расширить не может, — но там это требует громоздких записей и довольно серьезного знания  $\text{\TeX}$ 'овских внутренних механизмов; Спивак создал для этих целей удобные сокращенные обозначения (то есть макроопределения).

В 1984 году Лесли Лэмпорт (Leslie Lamport) создал макропакет  $\text{\LaTeX}$  (в отличие от  $\mathcal{A}\mathcal{M}\mathcal{S}\text{\TeX}$ 'а,  $\text{\LaTeX}$  дополнением к Plain  $\text{\TeX}$ 'у уже не являлся). Заключительная версия лэмпортовского макропакета, вышедшая в 1989 году, называлась  $\text{\LaTeX}$  2.09. Одной из важных новых черт  $\text{\LaTeX}$ 'а явилась возможность автоматической нумерации и — главное — автоматической генерации ссылок с помощью команд `\label`, `\ref` и `\pageref`.

Наконец, в 1995 году появилась серьезно переработанная версия  $\text{\LaTeX}$ 'а (она называлась  $\text{\LaTeX}$  2 $\epsilon$ ), описанию которой и посвящена эта книга. Одним из первоначальных толчков к переработке  $\text{\LaTeX}$ 'а было желание включить в него возможности  $\mathcal{A}\mathcal{M}\mathcal{S}\text{\TeX}$ 'а (в том числе шрифтовые). После первоначального периода быстрых изменений  $\text{\LaTeX}$  2 $\epsilon$  практически стабилизировался. В настоящее время под  $\text{\LaTeX}$ 'ом обычно понимают именно  $\text{\LaTeX}$  2 $\epsilon$ , и подавляющее большинство современных пользователей  $\text{\TeX}$ 'а пользуются именно  $\text{\LaTeX}$ 'ом, хотя некоторые математики старшего поколения сохраняют верность  $\mathcal{A}\mathcal{M}\mathcal{S}\text{\TeX}$ 'у.

Отметим еще, что исходные тексты макропакетов переносимы между платформами (Windows, Linux и т. п.). Более того, и сама программа  $\text{\TeX}$  сравнительно легко приспосабливается к различным типам компьютеров и операционных систем, и эти реализации действительно работают одинаково (Кнут разработал язык Web и написал программу на нем именно ради такой переносимости). Кстати, ради единообразия в программе  $\text{\TeX}$  не используются вычисления с плавающей запятой: на разных компьютерах они могут дать разные результаты, а это, в свою очередь, может повлиять на верстку.

Разумеется, пользовательский интерфейс и интерфейс с операцион-

ной системой в разных реализациях все равно устроены по-разному, от этого уж никуда не деться.

### А.3. 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»), содержащий информацию о том, какие буквы из каких шрифтов и в каком месте страницы надо разместить. Формат этих файлов стандартизирован и не зависит ни от использованного макропакета, ни от реализации  $\TeX$ 'а. Этот формат также разработан Кнудом: его описанию посвящена одна из глав кнудовской книги с полным комментированным текстом программы  $\TeX$ . Здесь мы только отметим, что в `dvi`-файлах указаны лишь названия шрифтов и номера символов в шрифтах, но не их начертания, которые  $\TeX$  и не знает: с его точки зрения каждая буква — это просто прямоугольник (точнее, «блок» — см. подробности в гл. VII). Все данные о размерах букв в шрифте (и некоторые дополнительные тонкости, связанные с их взаимным расположением на печати)  $\TeX$  берет из специальных файлов с расширением `.tfm`; (формат этих файлов также разработан Кнудом).

Для описания начертаний букв Кнут разработал специальный язык программирования под названием METAFONT. Метафонтовские описания символов по семантике (но не по синтаксису!) несколько напоминают появившийся позднее язык Postscript (по крайней мере и там, и там начертания задаются кривыми Безье). Транслятор языка METAFONT (также, разумеется, разработанный Кнудом) генерирует из метафонтовского исходного текста растровое описание символов шрифта (попросту говоря — матрицы из черных и белых точек) в том разрешении, какое требуется (кроме того, при этом генерируется `tfm`-файл).

Первоначальный комплект шрифтов, описанных на METAFONT'е, создал также Дональд Кнут в сотрудничестве с Германом Цапфом (Hermann Zapf). Этот набор шрифтов (гарнитура) называется Computer Modern (все шрифты, использованные до сих пор в этой книге, в своей латинской части повторяют шрифты Computer Modern). Надо ли говорить, что помимо этого Кнут опубликовал три книги, содержащие подробное описание программы METAFONT, ее комментированный текст, а также METAFONT'овские описания шрифтов Computer Modern (в дополнение к упомянутому выше описанию  $\TeX$ 'а и комментированному тексту программы  $\TeX$ ).

Вернемся к тому, что происходит с `dvi`-файлом. Он обрабатывается с помощью программы, называемой `dvi`-драйвером, осуществляющей печать, показ текста на экране и т. п. (для разных устройств и разных

нужд есть разные драйверы);  $\text{dvi}$ -драйвер в процессе работы использует файлы с растровыми описаниями шрифтов, а если такого описания не находится, он запускает программу  $\text{METAFONT}$ , которая генерирует этот файл с нужным разрешением. Иногда  $\text{METAFONT}$  автоматически запускается даже в процессе обработки  $\text{tex}$ -файла, если встречается запрос на шрифт, для которого нет соответствующего  $\text{tfm}$ -файла.

В современных реализациях  $\TeX$ 'а вся эта «кухня», как водится, от конечного пользователя спрятана: если, например, щелкнуть мышью по иконке, изображающей  $\text{dvi}$ -файл, то запустится  $\text{dvi}$ -драйвер, отвечающий за просмотр, в окне просмотра можно, в свою очередь, выбрать пункт меню «печать», и т. п.

#### А.4. Современная картина

Выше было описано то, что можно было бы назвать «классическим  $\TeX$ 'ом». Сейчас мы расскажем, что изменилось в этой картине в наши дни.

Во-первых, была разработана программа  $\varepsilon$ - $\TeX$ , сохраняющая полную совместимость с классическим  $\TeX$ 'ом, но обладающую дополнительными возможностями, малозаметными рядовому пользователю, но облегчающими работу верстальщиков, а также программистов, пишущих стилевые пакеты. Подавляющее большинство отличий  $\varepsilon$ - $\TeX$ 'а от классического  $\TeX$ 'а касается тонкостей, в нашей книге не упоминавшихся (скажем все же для примера, что в  $\varepsilon$ - $\TeX$ 'овском  $\LaTeX$ 'е возросло максимальное количество счетчиков, которые можно определить с помощью `\setcounter`).

Далее, как мы уже отмечали в предисловии, стандартом *de facto* в представлении текстов, независимом от операционной системы, стал не формат  $\text{dvi}$ , претендовавший на это, но формат  $\text{pdf}$ . Для конвертации  $\text{dvi}$ -файлов в  $\text{pdf}$ -файлы были разработаны специальные программы, но во всех современных поставках  $\TeX$ 'а (основанных уже на  $\varepsilon$ - $\TeX$ 'е) появилась возможность получать  $\text{pdf}$ -файл напрямую из  $\text{tex}$ -файла. Если речь идет о  $\LaTeX$ 'е, то при работе в командной строке для этих целей можно воспользоваться программой `pdflatex`: если ваш  $\text{tex}$ -файл называется `text.tex`, то в командной строке надо написать

```
pdflatex text
```

(вместо `latex text`). В более новых версиях можно не пользоваться `pdflatex`'ом, но в преамбуле файла написать

```
\pdfoutput=1
```

(в классическом  $\TeX$ 'е переменная `\pdfoutput` не определена) с тем же результатом. Если пользоваться стандартным комплектом  $\TeX$ 'овских

шрифтов (в том числе и кириллических), то получающийся в результате pdf-файл будет допускать поиск текста и копирование текстовых фрагментов.

Второе серьезное изменение, происшедшее в наши дни, касается шрифтов. Кнут, видимо, предполагал, что параллельно с распространением  $\TeX$ 'а будут появляться все новые и новые шрифты, написанные на METAFONT'e. На практике, однако, распространение метафонтонских шрифтов от распространения  $\TeX$ 'а стало быстро отставать. На данный момент таких шрифтов относительно немного (а русских шрифтов просто-таки мало), и новые практически не появляются. Помимо того, что полиграфическое качество классических кнотовских шрифтов гарнитуры Computer Modern вызывает определенные нарекания, дело еще и в том, что метафонтонские шрифты (любые, не только Computer Modern) можно использовать только с  $\TeX$ 'ом (а, скажем, программы Microsoft Word или Adobe Illustrator работать с ними не смогут). Гораздо более распространены шрифты в форматах TrueType (например, большинство шрифтов в системе Windows), Type1 (они же «PostScript-шрифты») и OpenType.

Уже классический  $\TeX$ , и тем более  $\varepsilon$ - $\TeX$ , можно заставить работать и с такими шрифтами (и вообще с любыми): надо только создать для них tfm-файлы (в случае метафонтонских шрифтов они генерируются одновременно с самим шрифтом, в остальных случаях это отдельное мероприятие) и написать стилевой пакет, объясняющий  $\TeX$ 'у, какие именно шрифты следует вызывать по командам вроде `\normalfont`, `\bfseries` и т. п. Кроме того, понадобится так называемый map-файл, содержащий информацию о соответствии имени, под которым шрифт известен  $\TeX$ 'у (это имя совпадает с именем tfm-файла) и реального файла шрифта; в этом же файле могут содержаться инструкции о перекодировке или деформации шрифта. Для части популярных шрифтов вся эта работа по приспособлению шрифта к  $\LaTeX$ 'у уже проделана.

Между тем создание усовершенствованных вариантов  $\TeX$ 'а продолжалось: после  $\varepsilon$ - $\TeX$ 'а, с которым классический  $\TeX$  полностью совместим, появилась и программа  $\XTeX$ , отличающаяся от  $\TeX$ 'а более серьезным образом (вариант  $\LaTeX$ 'а, полученный на основе  $\XTeX$ 'а, называется  $\XLaTeX$ ). В отличие от классического  $\TeX$ 'а и  $\varepsilon$ - $\TeX$ 'а, программа  $\XTeX$  (и, стало быть,  $\XLaTeX$ ) всегда порождает файлы в формате pdf, а dvi-файлы генерировать вообще не умеет. Кроме того,  $\XTeX$  работает с tex-файлами в юникодной кодировке. Но главное преимущество  $\XTeX$ 'а и  $\XLaTeX$ 'а состоит в том, что в этой системе появились широкие возможности для работы со шрифтами.

Надо сказать, что современный шрифт устроен довольно сложно: например, он может содержать различные варианты начертания одной и

той же буквы, выбор конкретного из которых происходит в зависимости от окружающих букв (например, в арабском или еврейском письме) или произволом наборщика («стилистические сеты» в декоративных шрифтах). Другой пример — расположенные друг над другом диакритические знаки, как во вьетнамском языке. Реализовать эти возможности при помощи обычных 256-символьных шрифтов — непростая задача, причем потребовался бы также довольно сложный стилевой пакет, перекрывающийся между ними. Тем не менее все требующиеся инструкции уже содержатся в современных OpenType шрифтах. Даже если вам не придется иметь дело с набором на арабском или использовать богатый альтернативами рукописный шрифт, вы можете извлечь определенную пользу из  $\XeTeX$ 'а: он позволяет легко подключать любой установленный в операционной системе шрифт формата OpenType, не требуя для работы `tfm`-, `vf`-, `map`- и `enc`-файлов (речь идет о текстовых шрифтах, с математическими все, конечно, сложнее).

Подавляющее большинство  $\LaTeX$ 'овских пакетов будет работать и в  $\XeTeX$ 'е. Проблемы могут возникнуть у пакетов, использующих для графики инструкции языка PostScript или существенно опирающихся на стандартные шрифты (пакет `soul`). Пакет `fontspec` предоставляет команды высокого уровня для обращения к шрифтам и их инструкциям, а пакет `polyglossia` выполняет роль пакета `babel` (см. приложение II).

Вот пример:

```
\documentclass{book}
\usepackage[no-sscript]{xltextra}
\usepackage{polyglossia}
\setdefaultlanguage{russian}
\newfontfamily\russianfont[Script=Cyrillic]{Mistral}
\begin{document}
Пример использования шрифта в \XeLaTeX
\end{document}
```

*Пример использования шрифта в  $\XeTeX$*

## Приложение Б. Библиографии в $\text{\LaTeX}$ 'е: $\text{BibTeX}$

Мы исходим из того, что читатель этого приложения знаком с содержанием раздела IV.7.3 и тем самым уже знает, как оформлять список литературы вручную.

Одна из проблем с таким оформлением — правильно расставить знаки препинания и правильно выбрать шрифты для различных частей библиографического описания. Даже зная соответствующий стандарт, оформлять это вручную довольно мучительно. Программа  $\text{BibTeX}$ , поставляемая в комплекте с  $\text{\TeX}$ 'ом, позволяет упростить и (полу)автоматизировать этот процесс.

### Б.1. Классический $\text{BibTeX}$

Расскажем сначала, что делать в случае, если в списке литературы русские буквы не используются.

Работа с  $\text{BibTeX}$ 'ом устроена следующим образом. Во-первых, необходимо записать список источников, на которые вы ссылаетесь в своем тексте, в специальный файл с расширением `.bib` (он еще называется **bib-файлом**) — как он устроен, рассказано ниже. В этом файле для каждого источника в отдельном поле записаны автор, название журнала, статьи, год, номер и пр. При этом вместо использования окружения `thebibliography` действовать надо так.

Предположим, что ваш  $\text{\TeX}$ -файл называется `mytext.tex`, а ваш **bib-файл** называется `mybiblio.bib`. Тогда в вашем файле `mytext.tex` надо написать следующее:

```
\bibliographystyle{plain}
\bibliography{mybiblio}
```

(сразу скажем, что аргумент команды `\bibliographystyle` задает стиль оформления библиографии; ниже мы скажем пару слов о том, какие еще стили бывают). Написав это и сохранив файл, надо запустить  $\text{\LaTeX}$ . Он сообщит о большом количестве неизвестных ссылок (раз у вас есть библиография, то, надо полагать, в файле имеются команды `\cite`). В этом месте в игру вступает  $\text{BibTeX}$ : после запуска  $\text{\LaTeX}$ 'а надо в командной строке выполнить команду

```
bibtex mytext
```

(в общем случае вместо `mytext` надо написать имя вашего  $\text{\TeX}$ -файла без расширения `.tex`; если вы пользуетесь редактором, интегрированным с  $\text{\LaTeX}$ 'ом, то в нем, возможно, предусмотрена специальная кнопка или пункт меню для запуска  $\text{BibTeX}$ 'а). После запуска  $\text{BibTeX}$ 'а надо вновь



запустить  $\text{\LaTeX}$ . Сначала он вновь пожалуется на неизвестные ссылки, но после следующего запуска все встанет на свои места и появится оформленный по всем правилам список литературы.

Формат  $\text{\bib}$ -файла удобно объяснить на примере (рис. Б.1). Всякий  $\text{\bib}$ -файл состоит из *записей* (запись — это то, что начинается с имени вида @... и заключено в фигурные скобки); запись, в свою очередь, может содержать различные *поля* (например,  $\text{\YEAR}=\{1974\}$ ). Сразу скажем, что в именах записей и полей строчные и прописные буквы не различаются (в отличие от того, к чему вы успели уже привыкнуть при работе с  $\text{\TeX}$ 'ом).

Проще всего объяснить, как работает запись @comment: все, что идет в фигурных скобках после этого имени,  $\text{\BibTeX}$ 'ом игнорируется, эта запись предназначена для внесения в  $\text{\bib}$ -файл пометок, рассчитанных на чтение человеком. А вот все прочие типы записей в нашем примере имеют уже прямое отношение к библиографии. Первая содержательная запись имеет тип @article; так описывают журнальные статьи. Разберем, что там написано.

В первой же строке, сразу после @article{, стоит библиографическая метка Har; при ссылках в тексте на указанную работу Дж. Харриса вы будете писать \cite{Har} (при составлении библиографию вручную, с помощью thebibliography, то вы бы написали \bibitem{Har}); обратите внимание, что в записи @article (как и в любой другой библиографической записи) после библиографической метки должна стоять запятая.

Далее идут поля записи, также разделенные запятыми (после самого последнего поля запятую можно не ставить). Смысл большинства из них ясен из названия. Обратим внимание на некоторые тонкости.

В поле AUTHOR пишется сначала фамилия автора, затем следует поставить запятую, а после запятой — его имя (или имена) или инициалы. Далее, обратите внимание на поле TITLE: в нем в фигурные скобки взята первая буква фамилии Severi. Сделано это потому, что при обработке заглавия  $\text{\BibTeX}$  зачастую расставляет строчные и прописные буквы по своему усмотрению (обычно — оставляет первую букву прописной, а все остальные делает строчными); однако если символ или несколько символов заключены в фигурные скобки, то с ними такого рода преобразований не производится. Остальные поля в первой записи особых пояснений не требуют, кроме поля MRNUMBER — это отсылка к номеру данной статьи в базе данных MathSciNet (см. ниже).

Вторая запись в нашем  $\text{\bib}$ -файле относится к книге. Обратите внимание на поля PUBLISHER (название издательства) и ADDRESS (город или города, в которых издательство базируется). Поле NOTE, также присутствующее в этой записи, предназначено для дополнительной ин-

```

@comment{Статья в журнале}
@article {Har,
AUTHOR = {Harris, Joe},
TITLE = {On the {S}everi problem},
JOURNAL = {Invent. Math.},
YEAR = {1986},
VOLUME = {84}, NUMBER = {3},
PAGES = {445--461},
MRNUMBER = {837522 (87f:14012)},
}
@comment{Книга}
@book {AGbook,
AUTHOR = {Hartshorne, Robin},
TITLE = {Algebraic geometry},
NOTE = {Graduate Texts in Mathematics, No. 52},
PUBLISHER = {Springer-Verlag}, ADDRESS = {New York}, YEAR = {1977},
MRNUMBER = {0463157 (57 \#3116)},
}
@comment{Статья в сборнике}
@inproceedings {Kleiman,
AUTHOR = {Kleiman, Steven L.}, TITLE = {Tangency and duality},
BOOKTITLE = {Proceedings of the 1984 {V}ancouver
conference in algebraic geometry},
SERIES = {CMS Conf. Proc.}, VOLUME = {6}, PAGES = {163--225},
PUBLISHER = {Amer. Math. Soc.},
ADDRESS = {Providence, RI}, YEAR = {1986},
MRNUMBER = {846021 (87i:14046)},
}
@comment{Книга с несколькими авторами, входящая в серию}
@book {SGA7.2,
TITLE = {Groupes de monodromie en g\`eom\`etrie alg\`ebrique. {II}},
SERIES = {Lecture Notes in Mathematics, Vol. 340},
NOTE = {S{\`e}minaire de G{\`e}om{\`e}trie Alg{\`e}brique du
Bois-Marie 1967--1969 (SGA 7 II),
Dirig{\`e} par P. Deligne et N. Katz},
AUTHOR = {Deligne, P. and Katz, N.},
PUBLISHER = {Springer-Verlag}, ADDRESS = {Berlin}, YEAR = {1973},
MRNUMBER = {0354657 (50 \#7135)},
}
@comment{Препринт}
@unpublished{jonson,
AUTHOR = {Jonson, Jon},
TITLE = {On {J}acobian varieties of smooth rational curves},
NOTE = {Preprint \href{http://arxiv.org/abs/9999.9999}
{arXiv:9999.9999}[math.ZZ]},
}

```

Рис. Б.1. Пример bib-файла

формации, которую вы хотите внести в библиографическое описание. В данном случае сказано, что книга входит в серию «Graduate Texts in Mathematics» и что она имеет в этой серии номер 52. Можно было бы сообщить эту информацию и более упорядоченным способом, вместо NOTE написав

```
SERIES = {Graduate Texts in Mathematics},
NUMBER = {52},
```

Следующая запись представляет статью, опубликованную в сборнике докладов (proceedings) некоторой конференции. Обратите внимание на поле BOOKTITLE — это заглавие не статьи, а сборника в целом. Сам этот сборник также входит в какую-то серию («продолжающееся издание», как выражаются библиографы).

Следующая запись демонстрирует, как надо оформлять источники с несколькими авторами: их надо перечислять, разделяя служебным словом **and** (без запятых). Заодно мы видим, что поля внутри одной записи могут идти в произвольном порядке.

Наконец, последняя запись — это (на сей раз вымышленный) препринт из электронной библиотеки препринтов **arxiv.org**. Для всех неопубликованных текстов разумно использовать запись **@unpublished**: кроме автора и заглавия, в ней есть поле NOTE, в которой имеет смысл описать своими словами, каким образом текст обнародован и как его можно получить. Команда **\href**, предназначенная для создания гиперссылок в pdf-файле, будет определена при подключении пакета **hyperref**; см. приложение Г.

Вот как будет выглядеть наша библиография (если указать **plain** в качестве **\bibliographystyle**) после обработки  $\text{BibTeX}$ 'ом и  $\text{\LaTeX}$ 'ом.

- [1] P. Deligne and N. Katz. *Groupes de monodromie en géométrie algébrique. II. Lecture Notes in Mathematics*, Vol. 340. Springer-Verlag, Berlin, 1973. Séminaire de Géométrie Algébrique du Bois-Marie 1967–1969 (SGA 7 II), Dirigé par P. Deligne et N. Katz.
- [2] Joe Harris. On the Severi problem. *Invent. Math.*, 84(3):445–461, 1986.
- [3] Robin Hartshorne. *Algebraic geometry*. Springer-Verlag, New York, 1977. Graduate Texts in Mathematics, No. 52.
- [4] Jon Jonson. On Jacobian varieties of smooth rational curves. Preprint arXiv:9999.9999 [math.ZZ].
- [5] Steven L. Kleiman. Tangency and duality. In *Proceedings of the 1984 Vancouver conference in algebraic geometry*, volume 6 of *CMS Conf. Proc.*, pages 163–225, Providence, RI, 1986. Amer. Math. Soc.

Обратите внимание, что Bib $\TeX$  отсортировал наши источники в алфавитном порядке авторов. Заметьте также, что поля `MRNUMBER` в библиографии никак не отражены: если обработка какого-то поля в выбранном вами стиле оформления библиографии не предусмотрена, то Bib $\TeX$  это поле молча игнорирует.

Вот, для сравнения, что получится, если в качестве стиля оформления библиографии вместо `plain` указать `abbrv` (другой популярный стиль):

- [1] P. Deligne and N. Katz. *Groupes de monodromie en géométrie algébrique. II. Lecture Notes in Mathematics*, Vol. 340. Springer-Verlag, Berlin, 1973. Séminaire de Géométrie Algébrique du Bois-Marie 1967–1969 (SGA 7 II), Dirigé par P. Deligne et N. Katz.
- [2] J. Harris. On the Severi problem. *Invent. Math.*, 84(3):445–461, 1986.
- [3] R. Hartshorne. *Algebraic geometry*. Springer-Verlag, New York, 1977. Graduate Texts in Mathematics, No. 52.
- [4] J. Jonson. On Jacobian varieties of smooth rational curves. Preprint arXiv:9999.9999 [math.ZZ].
- [5] S. L. Kleiman. Tangency and duality. In *Proceedings of the 1984 Vancouver conference in algebraic geometry*, volume 6 of *CMS Conf. Proc.*, pages 163–225, Providence, RI, 1986. Amer. Math. Soc.

Разница в том, что на сей раз Bib $\TeX$  сократил имена авторов до инициалов (в тех случаях, когда они не были уже сокращены в `bib`-файле).

А вот что получится со стилем `amsalpha`, разработанным Американским математическим обществом:

- [DK73] P. Deligne and N. Katz, *Groupes de monodromie en géométrie algébrique. II*, Lecture Notes in Mathematics, Vol. 340, Springer-Verlag, Berlin, 1973, Séminaire de Géométrie Algébrique du Bois-Marie 1967–1969 (SGA 7 II), Dirigé par P. Deligne et N. Katz. MR 0354657 (50 #7135)
- [Har77] Robin Hartshorne, *Algebraic geometry*, Springer-Verlag, New York, 1977, Graduate Texts in Mathematics, No. 52. MR 0463157 (57 #3116)
- [Har86] Joe Harris, *On the Severi problem*, *Invent. Math.* **84** (1986), no. 3, 445–461. MR 837522 (87f:14012)
- [Jon] Jon Jonson, *On Jacobian varieties of smooth rational curves*, Preprint arXiv:9999.9999 [math.ZZ].

- [Kle86] Steven L. Kleiman, *Tangency and duality*, Proceedings of the 1984 Vancouver conference in algebraic geometry (Providence, RI), CMS Conf. Proc., vol. 6, Amer. Math. Soc., 1986, pp. 163–225. MR 846021 (87i:14046)

Как видите, на сей раз поля `MRNUMBER` также пошли в дело. Обратите также внимание, что изменился и порядок, в котором расположены в библиографии источники: стиль `amsalpha` обозначает их не номерами, а сокращениями из фамилии и номера публикации, а затем сортирует в алфавитном порядке эти сокращения; в результате книга Хартсхорна (1977 год) оказалась по алфавиту раньше статьи Харриса (1986).

В стандартную поставку  $\text{\LaTeX}$ ’а обычно входят и другие библиографические стили, помимо трех упомянутых. Как они называются, нетрудно узнать в интернете, а в каком виде они представляют библиографию, можно выяснить, поэкспериментировав с ними.

Что бы ни было записано в вашем `bbl`-файле,  $\text{\BibTeX}$  по умолчанию включает в список литературы только те источники, на которые вы ссылаетесь с помощью команды `\cite`. Если вам все-таки хочется, чтобы в список литературы вошла работа, ссылка на которую в тексте отсутствует, следует воспользоваться командой `\nocite`: если, скажем, написать (в преамбуле или непосредственно перед командой `\bibliography`) `\nocite{bs2013}`, то источник, обозначенный в `bib`-файле как `bs2013`, войдет в библиографию независимо от того, есть ли на него ссылка в тексте. В аргументе команды `\nocite` можно перечислить через запятую несколько обозначений для ваших библиографических источников; можно, наконец, в качестве аргумента этой команды поставить звездочку: если написать `\nocite{*}`, то в библиографию войдет все содержимое `bib`-файла.

Надо еще объяснить, откуда, собственно говоря, брать данные для `bib`-файла. Можно написать этот файл самому и пополнять по мере необходимости, но если вы математик, то удобнее воспользоваться поддерживаемой Американским математическим обществом базой данных MathSciNet, в которой собраны рефераты на практически все существующие работы по математике (точнее говоря, на все послевоенные и на некоторое количество часто цитируемых более ранних работ) вкупе с готовыми записями для `bib`-файла. Доступ к этой базе данных платный, но в крупных университетах и академических институтах (в том числе и в России) он обычно оплачен, так что можно добраться до нее с рабочих компьютеров. Если доступа к MathSciNet’у у вас нет, можно воспользоваться бесплатной базой данных MrLookup, которую также поддерживает Американское математическое общество; в отличие от MathSciNet’а, эта база выдает не более трех ответов на любой запрос,

так что для нахождения нужного библиографического описания порой требуется некоторая изобретательность.

Другая библиографическая база данных, аналогичная MathSciNet'у, называется Zentralblatt Mathematik. Доступ к ней в полном объеме также предоставляется только за деньги, но даже при отсутствии подписки Zentralblatt показывает часть ответов на ваш запрос (не более трех), так что если запрос достаточно узок, можно найти требуемое и бесплатно. В базе Zentralblatt лучше, чем в MathSciNet, отражены довоенные работы немецких математиков (но менее корректно прописаны `bib`-файлы).

## Б.2. $\text{\BibTeX}$ и русские буквы

Все сказанное выше относится исключительно к случаю, когда библиографию надо оформить по правилам, принятым на Западе, плюс (что еще важнее) все записи в `bib`-файле написаны латиницей. Если попытаться применить стандартный  $\text{\BibTeX}$  с его стандартными стилями к `bib`-файлам, содержащим кириллицу, то ничего хорошего не выйдет по двум причинам: во-первых,  $\text{\BibTeX}$  в его исходном виде не сможет корректно отсортировать русские слова и установить соответствие между строчными и прописными буквами, во-вторых, западные и отечественные правила оформления библиографии сильно отличаются.

Для борьбы с первой из этих трудностей в современном  $\text{\BibTeX}$ 'е предусмотрена возможность при запуске указывать ссылку на специальный файл, в котором задается порядок сортировки и соответствие между строчными и прописными буквами. Такие файлы имеют расширение `.csf`; мы будем называть их `csf`-файлами. При запуске программы  $\text{\BibTeX}$  можно указать ей, что правила сортировки надо взять из соответствующего `csf`-файла. Например, если вы хотите создать библиографию для файла `mytext.tex` с правилами сортировки и соответствия «прописные-строчные», заданными в файле `cp1251.csf`, то надо сказать

```
bibtex --huge --csfile cp1251.csf mytext
```

(ключ `--huge` рекомендуется указывать, чтобы  $\text{\BibTeX}$  не сломался при работе с файлами в восьмибитной кодировке).

Разумеется, для каждой кодировки русских букв нужен свой `csf`-файл. Если вы раздобыли его не для той кодировки, которой пользуетесь, его придется модифицировать самостоятельно. Это нетрудно: `csf`-файлы — текстовые файлы достаточно простого формата.

Что касается второй трудности — оформления библиографии, пусть и правильно отсортированной, в соответствии с отечественными стандартами, — то для ее преодоления необходимы соответствующие стили

(то, что задается в аргументе команды `\bibliographystyle`). Такие стили задаются в файлах с расширением `.bst` (`bst`-файлах). Соответствующие файлы разработаны; в некоторые поставки  $\text{\LaTeX}$ ’а они уже входят, и в любом случае они доступны на сайте CTAN (см. гл. VIII) — ищите их там по ключевому слову `gost`. Формат `bst`-файлов сложен (по сути дела, это довольно богатый по возможностям язык программирования); его полное описание также доступно в интернете (по состоянию на весну 2014 года имелось руководство под эффектным заглавием «Tame the beast»).

Разумеется, после того как вы раздобыли или разработали `csf`- или `bst`-файлы, отвечающие вашим нуждам, надо предпринять еще некоторые действия, чтобы  $\text{Bib}\text{\LaTeX}$  научился их «видеть». См. приложение О.

## Приложение Г. Гиперссылки в pdf-файлах

В pdf-файлы, генерируемых из tex-файлов, можно вставлять «кликабельные» ссылки на web-страницы. Для этого проще всего воспользоваться стилевым пакетом `hyperref`. У этого пакета имеется масса возможностей, полное описание которых можно найти в оригинальной документации. Мы расскажем только о наиболее насущном.

Если вы хотите получить в pdf-файле точный адрес какого-нибудь интернет-ресурса (то есть URL), обладающий тем свойством, что при щелчке мышью по этому адресу ваш браузер откроет соответствующую страницу, то надо воспользоваться командой `\url` — эта команда с одним аргументом, в котором нужный нам URL и записывается. Например, можно написать `\url{http://google.com}`.

Как известно, в адресах интернет-ресурсов могут встречаться символы `~`, `%` и `#`. Так вот, в аргументе команды `\url` их следует записывать непосредственно, без предосторожностей: в таком контексте они не будут восприняты как Т<sub>Е</sub>X’овские спецзнаки. Более того, адрес, записанный с помощью команды `\url`, сможет при необходимости быть автоматически перенесен в разумных местах (на точках и символах `/`) — полезное свойство, если учесть, что адреса нередко бывают длинными.

Чтобы получить в pdf-файле текст, при щелчке по которому происходит переход по данному адресу, надо воспользоваться командой `\href` с двумя аргументами: в первом записывается адрес, во втором — «кликабельный» текст. Пример:

```
\href{http://vk.com}{популярная социальная сеть}
```

Адрес в первом аргументе команды `\href` следует записывать непосредственно и предосторожностей с символами `~`, `%` и `#` не предпринимать — как и в случае с аргументом команды `\url`.

При подключении пакета `hyperref` появляется не только возможность создавать «кликабельные» ссылки на внешние ресурсы: при щелчке мышью по любому номеру внутри файла, полученному с помощью команды `\ref` или `\pageref`, будет происходить переход на указанную страницу или к указанному месту в файле. Более того, соответствующие места в pdf-файле будут выделены.

Наконец (опять-таки при подключении `hyperref`), для разделов текста, оформленных по стандартным Л<sub>А</sub>T<sub>Е</sub>X’овским правилам (с помощью команд `\chapter`, `\section` и пр.), в pdf-файле создаются закладки (bookmarks), щелкнув по которым, можно перейти к соответствующему разделу. С точки зрения пользователя с ними все аналогично оглавлению: при каждом запуске pdf<sub>l</sub>atex’а информация о закладках записывается в специальный файл (с расширением `out`), при каждом из



последующих запусков она из него считывается (так что после самого первого запуска закладки не появятся). В закладках отражаются те же разделы документа, которые отражаются в оглавлении.

Если ваш текст написан на русском языке и вы его оформляете с помощью пакетов `inputenc` и `babel`, как объяснялось в разд. IV.1, то русские названия разделов документа будут корректно передаваться в закладках; к сожалению, если воспользоваться весьма привлекательным в других отношениях способом работы с русскими текстами, описанным в разд. II.5 приложения II, то вместо русских букв вы увидите в закладках невесть что.

## Приложение Д. Диаграммы (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 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, как водится, почитать в оригинальной документации).

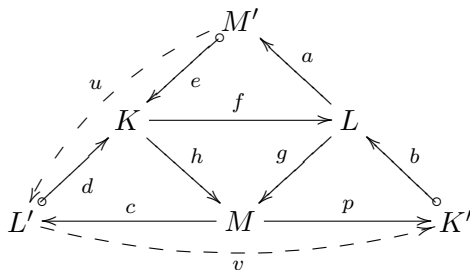
### Д.1. Пример с комментариями

Чтобы можно было пользоваться пакетом `Xy-pic` для набора коммутативных диаграмм, надо подключить стилевой пакет `xu` с опцией `all`. Иными словами, надо написать в преамбуле

```
\usepackage[all]{xy}
```

Объяснить, как набирать с помощью `Xy-pic`'а коммутативные диаграммы, удобно на примере.

Рассмотрим следующую диаграмму:



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`.

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  $5 \times 3$ :  $M'$  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 будет 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, `[ddl]` 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  $\wedge$  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.

## Д.2. Некоторые общие правила

Синтаксис Xy-pic'a весьма сложен, и мы не будем пытаться изложить его полностью (автор его в полном объеме и не знает). Вместо этого приведем несколько приемов, наиболее, на наш взгляд, полезных на практике.

## Д.3. 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  $\wedge$  или  $\_$  и надписью знак  $-$  (минус):

$$A \times B \times C \xrightarrow{-f_1} D$$

```


$$\begin{array}{l} \text{\$}\xymatrix{\text{A}\times B\times C\ar@{.>}[r]^-{f_1}\text{\&D}\text{\$}} \end{array}$$


```

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  $\wedge$  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):

$$A \xrightarrow{f} B$$

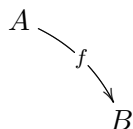
```


$$\begin{array}{l} \text{\$}\xymatrix{\text{A}\ar[r]^{(.25){f}}\text{\&\&B}\text{\$}} \end{array}$$


```

Заодно мы применили конструкцию, рекомендуемую авторами пакета для ситуаций, когда вся «диаграмма» укладывается в одну строчку: поместили  $\text{\@1}$  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  $\wedge$  или  $\_$  написать  $|$ , как в следующем примере:



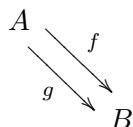
```

 $\xymatrix{$ 
 $A \ar@{~}[rd] | f \\\$ 
 $\&B$ 
 $\}$ 

```

#### Д.4. Сдвинутые стрелки

Стрелки можно сдвигать параллельно себе. Для этого используется конструкция  $\ar@<...>$ , где на месте точек стоит длина в Т<sub>Е</sub>X'овских единицах, указывающая величину сдвига. Если эта длина положительна, то сдвиг будет влево (если смотреть от начала к концу стрелки), если отрицательна, то вправо:



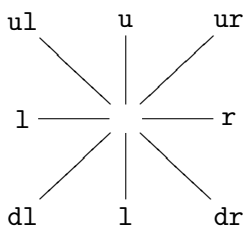
```

 $\xymatrix{$ 
 $A \ar@{<1ex>[dr] ^f$ 
 $\ar@{<-1ex>[dr] _g \\\$ 
 $\&B$ 
 $\}$ 

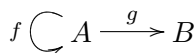
```

#### Д.5. Еще об изогнутых стрелках

Кроме конструкции  $\ar@{...}/$ , изгиб стрелки можно задавать конструкцией  $\ar@{направление\_выхода, направление\_входа}$ , где *направление\\_выхода* и *направление\\_входа* — это буквы или пары букв, значение которых показано на следующем рисунке:



Пример:



```

 $\xymatrix{$ 
 $A \ar@{(ul,dl)} [] _f \ar[r] ^g \&B$ 
 $\}$ 

```

Заодно показано, как с помощью «пустого» указателя направления  $[]$  и  $\ar@{...}$ -конструкции можно напечатать стрелку, ведущую из формулы в нее же.

## Д.6. 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.

$-->$	<code>@{--&gt;}</code>	$\Longrightarrow$	<code>@{=&gt;}</code>	$\cdots\cdots\cdots>$	<code>@{.&gt;}</code>
$\longrightarrow$	<code>@{&gt;-&gt;}</code>	$\Longrightarrow$	<code>@3{-&gt;}</code>	$\longleftrightarrow$	<code>@{&lt;-&gt;}</code>
$\longrightarrow$	<code>@{-&gt;&gt;}</code>	$\longleftrightarrow$	<code>@{&lt;=&gt;}</code>	$\longmapsto$	<code>@{ -&gt;}</code>
$\hookrightarrow$	<code>@{~{(-&gt;}</code>	$\longrightarrow$	<code>@{-~{&gt;}}</code>	$\longrightarrow$	<code>@{-_{&gt;}}</code>
$\longrightarrow$	<code>@{-}</code>	$\Longrightarrow$	<code>@{=}</code>	$\Longrightarrow$	<code>@3{-}</code>

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.

## Д.7. Оптимизация и предупреждение ошибок

Пакет *Xy-pic* заставляет *TeX* работать буквально на пределе возможностей; файлы, в которых используются *Xy-pic*'овские конструкции, обрабатываются относительно 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. Впрочем, возможно, что при нынешних компьютерных мощностях такая оптимизация не слишком нужна.

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 использовании *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 Т<sub>E</sub>X'ническим причинам эти макросы могут не сработать.

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 воспользоваться программой Метапост (см. приложение М).

## Приложение И. Интернационализация $\text{\LaTeX}$ 'а

Под интернационализацией в компьютерном мире обычно понимается приспособление системы к работе с различными (человеческими) языками. Посмотрим, как эти вопросы решаются в  $\text{\LaTeX}$ 'е.

Таблица И.1. Кодировка OT1

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	˘	˚	ˇ	˘	˘	˙	˚	˚	I	<	>
"10	“	”	^	˘	˘	—	—		o	i	j	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	{		}	~	-

### И.1. Внутренняя кодировка шрифта

Как мы уже писали в разд. III.5.4, каждый шрифт с точки зрения  $\text{\LaTeX}$ 'а характеризуется гарнитурой (общим стилем начертания букв), семейством, размером, насыщенностью и прочими аналогичными атрибутами. Все это, спору нет, важно, но еще важнее знать, какие, собственно говоря, символы представлены в этом шрифте, а также в каком порядке они там расположены (выражаясь более корректно — какие у них коды). Вот эта информация и называется в  $\text{\LaTeX}$ 'е кодировкой шрифта.

В ранних версиях  $\text{\TeX}$ 'а использовались только «семибитные» шрифты (из 128 символов). На табл. И.1 представлен один из классических  $\text{\TeX}$ 'овских шрифтов, иллюстрирующий так называемую кодировку OT1.

(К обозначениям: скажем, на пересечении строки "40 и столбца "0C стоит символ с шестнадцатеричным кодом 4C, т. е. десятичным 76.)

Коды латинских букв в этой таблице совпадают с так называемыми ASCII-кодами, т. е. с номерами этих символов в любой кодовой таблице (code page), используемой современными операционными системами.

Обратите еще внимание на лигатуры ff, fi и пр. во второй строке, а также символы для диакритических знаков в первых двух строках.

Начиная с версии 3.0 (1989 год) в  $\text{\TeX}$ 'е предусмотрена возможность работы с восьмибитными шрифтами (а также со входными файлами в восьмибитной кодировке). Соответственно, были разработаны (не в 1989 году, а позднее) и кодировки для восьмибитных шрифтов. На табл. И.2



Таблица И.2. Кодировка T1

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	˜	°	˘	˙	˚	˛	˜	ˆ	˜	˘	˙
"10	“	”	„	«	»	—	—		o	ı	J	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	Ā	Ą	Ć	Č	Ď	Ě	Ě	Ğ	Ĺ	Ł	Ł	Ń	Ň	Ŋ	Ō	Ŕ
"90	Ř	Ś	Š	Ş	Ť	Ţ	Ũ	Ū	Ÿ	Ž	Ž	Ž	IJ	İ	đ	§
"A0	ă	ą	ć	č	ď	ě	ę	ğ	ĺ	ł	ł	ń	ň	ŋ	ó	í
"B0	ř	ś	š	ş	ť	ţ	û	û	ÿ	ž	ž	ž	ij	ı	ı	£
"C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
"D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	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	œ	ø	ù	ú	û	ü	ý	þ	ß	ß

представлена так называемая кодировка T1, содержащая в верхней половине готовые символы для букв с диакритическими знаками из ряда наиболее распространенных европейских языков.

В своей «нижней» половине (символы с кодами от 0 до 127) она близка к кодировке OT1, хотя совпадает с ней не полностью.

Шрифты, представленные в этих таблицах, относятся к гарнитуре Computer Modern и имеют прямое светлое начертание. Можно было бы продемонстрировать аналогичные таблицы с теми же кодировками, но для других шрифтов, и почти всегда на тех же местах будут стоять «те же» символы<sup>1</sup>.

На табл. И.3 представлена кодировка T2A, у которой в позициях от 0 до 127 расположены те же символы, что в кодировке T1, а в «верхней» половине (от 128 до 255) — кириллица, включая дополнительные буквы и буквы с диакритическими знаками, использующиеся в пись-

<sup>1</sup>Провести этот принцип без «почти» все же не удастся: что, например, делать, если в одной гарнитуре лигатура ffi есть (как в Computer Modern в позиции с шестнадцатеричным кодом 1E), а в другой она не предусмотрена?

Таблица И.3. Кодировка T2A

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	˘	°	˘	˘	–	·	˘	˘	I	<	>
"10	“	”	^	˘	˘	–	–		o	l	j	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	Г	Г	Ђ	Ђ	ђ	Ж	З	Љ	Ї	К	К	К	Æ	Њ	Н	Ѕ
"90	Ө	Ѓ	Ў	У	У	Х	Ц	Ч	Ч	€	Э	Њ	Ё	№	¤	§
"A0	г	г	ђ	ђ	ђ	ж	з	љ	ї	к	к	к	æ	њ	н	ѕ
"B0	ө	ѓ	ў	у	у	х	ц	ч	ч	€	э	њ	ё	„	«	»
"C0	A	B	B	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
"D0	P	C	T	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
"E0	a	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
"F0	p	c	t	y	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

менностях на кириллической основе. Впрочем, в одну таблицу все эти дополнительные символы все равно не помещаются, так что наряду с кодировкой T2A имеются еще T2B и T2C, отличающиеся от T2A только дополнительными кириллическими символами (список кодировок из стандартной  $\text{\LaTeX}$ 'овской поставки перечисленными не исчерпывается).

Как же, однако,  $\text{\LaTeX}$  узнает о том, шрифты какой кодировки (и вообще — какие шрифты) ему надо использовать в тексте? По умолчанию, если никаких действий не предпринимать (например написать `\documentclass{article}`, а после этого — сразу `\begin{document}`),  $\text{\LaTeX}$  будет считать, что в тексте используется шрифт в кодировке OT1 (гарнитуры Computer Modern, прямой светлый с засечками, но не об этом сейчас речь). Так как кодировка OT1 семибитная, символы с кодом больше 127 будут  $\text{\LaTeX}$ 'ом проигнорированы (неважно, записаны они в `tex`-файле напрямую или мы пытаемся добраться до них с помощью команды `\symbol`). Если текст написан исключительно по-английски и автор готов смириться с тем, что в (весьма редко встречающихся) ан-

глийских словах с диакритическими знаками автоматический перенос будет невозможен, то большего и не требуется.

Пусть, однако, нам нужен именно восьмибитный шрифт — например, мы пишем по-русски или на языке, использующем письменность на латинской основе с диакритикой (чтобы француз для набора слова *théâtre* мог так и написать в *tex*-файле *théâtre*, а не *th\’e\^atre*). Самый непосредственный способ сообщить ЛАТЭХ’у, в какой именно кодировке вам нужны шрифты — воспользоваться стилевым пакетом *fontenc*. Именно, если в преамбуле написать

```
\usepackage[кодировка]{fontenc}
```

где *кодировка* — обозначение для кодировки шрифта (OT1, T1, T2A и т. п.), то после этого шрифтом по умолчанию станет шрифт, использующий указанную вами кодировку. В качестве стилиевой опции для *fontenc* можно указать не одну кодировку, а несколько — тогда кодировкой по умолчанию станет указанная последней, но ЛАТЭХ будет знать, что в тексте могут использоваться и кодировки, указанные в списке опций пакет *fontenc* ранее. Если, скажем, в преамбуле написано

```
\usepackage[T1,T2A]{fontenc}
```

то кодировкой по умолчанию будет «русская» кодировка T2A, но если сказать `\fontencoding{T1}\selectfont` (без `\selectfont` не работает!), то после этого ЛАТЭХ начнет воспринимать символы из *tex*-файла как относящиеся к шрифтам «западноевропейской» кодировки T1:

И È	<code>\symbol{"C8}</code>
	<code>\fontencoding{T1}\selectfont</code>
	<code>\symbol{"C8}</code>

(загляните в таблицы и убедитесь, что в кодировке T1 шестнадцатеричному коду C8 соответствует буква È, а в кодировке T2A — буква И).

Если команды для смены кодировки (`\fontencoding` и `\selectfont`) дать внутри группы, то по выходе из группы восстановится прежняя кодировка.

На самом деле некоторые кодировки (в частности, OT1 и T1) известны ЛАТЭХ’у заранее, даже если не подключать пакет *fontenc* с соответствующими опциями, но если пакет все же подключить и опции указать, то ничего плохого не будет.

## И.2. Соответствие внешней и внутренней кодировок.

### Часть 1

Итак, мы теперь знакомы с понятием «кодировка ЛАТЭХ’овского шрифта». Вообще говоря, эта кодировка (в своей «верхней половине» — сим-

волы с кодами  $> 127$ ) не совпадает с кодировкой букв в `tex`-файлах. Как же задается соответствие между ними?

Проще всего тем, кто пишет тексты на каком-то из наиболее распространенных западноевропейских языков (немецком, французском и др.). Именно, в ЛАТЭХ'овских шрифтах с кодировкой T1 коды используемых в этих языках букв с диакритиками и дополнительных букв совпадают с кодами в так называемой кодовой таблице `cp1252`, так что, сообщив ЛАТЭХ'у, что используются шрифты с кодировкой T1 (см. предыдущий раздел), далее можно непосредственно вводить соответствующие символы с клавиатуры. Если даже ваш компьютер на такой ввод не настроен, можно, пользуясь командами из разд. III.4, набирать, скажем, â как `\^a` — при использовании шрифтов в кодировке T1 в слове, содержащем эту букву с диакритикой, будет сохраняться возможность автоматического переноса, как если бы при наборе мы написали â. (Это относится именно к кодировке T1; для большинства других кодировок, в частности, для основной русской кодировки T2A, такой сервис не предусмотрен.)

Правда, возможность автоматического переноса — не то же самое, что правильные автоматические переносы: по умолчанию таблицы переноса в ТЭХ'е предназначены для английского языка, и для языков, отличных от английского, могут приводить к некорректным результатам. Чтобы переносы были не только возможными, но и правильными, надо дополнительно подключить пакет `babel` (см. разд. II.4).

С русскими текстами все сложнее. В настоящее время, как известно, для русских букв используются как минимум три разные восьмибитные кодировки: `cp1251` («виндовая»), `cp866` («досовская», она же «альтернативная») и `koï8-r` (используемая в UNIX-подобных системах).

Проще всего, если ваш текстовый редактор сохраняет файл в кодировке `cp1251` (таково поведение по умолчанию большинства редакторов под Windows). Дело в том, что коды всех русских букв, кроме ё, в кодовой таблице `cp1251` и в ЛАТЭХ'овской внутренней кодировке T2A совпадают. Поэтому если сохранить файл в кодировке `cp1251`, то русские буквы можно набирать непосредственно, с тем исключением, что ё и Ё надо будет набирать как `\"е` и `\"Е` соответственно. Разумеется, мы тут подразумеваем, что в преамбуле написано

```
\usepackage[T2A]{fontenc}
```

Если вы набираете русские тексты в кодировке `cp866` или `koï8-r`, то этот номер уже не пройдет: надо каким-то образом объяснить ТЭХ'у, что, скажем, код буквы ы в шрифте должен быть равен не 235, как в кодировке `cp866`, и не 217, как в кодировке `koï8-r`, но 251, как во внут-

ренней ЛАТЭХ'овской кодировке Т2А. Тут есть два принципиально разных подхода.

Первый вариант состоит в том, что во всех современных реализациях ТЭХ'а предусмотрена возможность перекодирования «на лету» входных файлов из той кодировки, в которой они записаны на диске, в нужную нам внутреннюю кодировку шрифта. Таблица соответствия записывается в специальном файле с расширением `tsx`. Если, например, наш `tsx`-файл задает перекодировку из «альтернативной» (ср866) кодировки `tex`-файла в кодировку Т2А (такой файл естественно назвать `alt2a.tsx`), то вызов ЛАТЭХ'а с учетом такой перекодировки выглядит так:

```
latex --translate-file alt2a.tsx имя_файла
```

Формат `tsx`-файлов очень прост. Это текстовый файл, в каждой строке которого через пробел записаны два числа: сначала код символа из `tex`-файла, затем код, который должен вместо него увидеть ТЭХ. Коды символов можно записывать либо в десятичной системе (как есть), либо в восьмеричной (тогда они должны начинаться с нуля), либо в шестнадцатеричной (тогда они должны начинаться с 0x). Пустые строки игнорируются, знак процента означает, как и в ТЭХ'е, комментарий. Например, `tsx`-файл, задающий перекодировку из ср866 в OT2, мог бы начинаться так:

```
0x80 0xC0 % буква А
```

```
0x81 0xC1 % буква Б
```

```
.....
```

А если вы записываете свои `tex`-файлы в «виндовой» кодировке ср1251 и при этом хотите набирать буквы ё и Ё непосредственно, а не как `\"e` и `\"E`, то можно использовать `tsx`-файл из двух строчек:

```
0xA8 0x9C
```

```
0xB8 0xBC
```

(B8 — шестнадцатеричный код буквы ё в кодировке ср1251, BC — код этого же символа в Т2А).

Если ТЭХ запускается с перекодировкой, заданной `tsx`-файлом, то во всех файлах, которые ТЭХ в процессе трансляции записывает на диск, происходит обратная перекодировка. В частности, становятся нормально читаемыми `log`-файл, `toc`-файл с оглавлением, `idx`-файл с сырьем для индекса и пр.

Как ни странно, `tsx`-файлы для перекодировки ср866 или koi8-r в кодировку Т2А есть не во всех поставках ТЭХ'а. Как объяснялось выше, такой файл очень просто написать самому. Кроме того, готовые файлы `win2t2.tsx`, `koi2t2.tsx` и `alt2t2.tsx` легко находятся в интернете.

Заключительное замечание: чтобы не писать все время в командной строке `--translate-file=...`, разумно написать однострочный `bat`-файл (если вы работаете под Windows) или определить `alias` (в UNIX-подобных системах).

### И.3. Соответствие внешней и внутренней кодировок.

#### Часть 2

Второй способ установить нетривиальное соответствие между кодировкой символов в `tex`-файле и кодировкой ЛАТЭХ'овского шрифта достигается средствами самого ТЭХ'а: он состоит в использовании пакета `inputenc`. Как использовать этот пакет для русского языка, рассказано на с. 89 (и повторено в разд. IV.1): надо в преамбуле вызвать этот пакет с опцией, соответствующей используемой вами русской кодировке.

Тогда никаких хитростей с `tsx`-файлами вам не потребуется, ЛАТЭХ можно будет вызывать непосредственно, но за это удобство придется заплатить довольно дорогую цену: ни в том, что выводится на экран, ни в `toc`-файлах, ни в `idx`-файлах русских букв видно не будет: вместо этого будет что-нибудь вроде `\cyrshch` или `\CYRP`. В результате выдача на экран будет нечитаемой, `log`- и `toc`-файлы — тоже, а составлять индекс будет невозможно без серьезных дополнительных усилий. С другой стороны, если вы используете пакет `hyperref`, то при этом будут корректно отображаться русские закладки в `pdf`-файлах (см. приложение Г), тогда как при русификации с помощью `tsx`-файлов это не получится. Пользоваться ли пакетом `inputenc` — решать вам.

### И.4. Пакет `babel`

Итак, тем или иным способом мы научили ЛАТЭХ правильно понимать символы из «верхней половины» кодовой таблицы. Чтобы корректно обрабатывать текст на языке, отличном от английского, это необходимо, но недостаточно. Как минимум, надо еще загрузить таблицу переносов, специфическую для используемого языка, и сделать так, чтобы на нужном вам языке, а не по-английски, печатались стандартные заголовки вроде «Глава», «Рис.», «Приложение» и т. п. (см. разд IV.5.2). Кроме того, полиграфические традиции в разных языках тоже различаются: например, в французских текстах принято отделять запятую от предшествующего слова небольшим пробелом (и хочется, чтобы такие пробелы расставлялись автоматически), рекомендуемые размеры тире в русских текстах отличаются от английских — и —, и т. п. Все эти задачи призван решить стилевой пакет `babel` (название — от Вавилонской башни).

У этого пакета имеется множество стилевых опций, названия которых обычно совпадают с названиями соответствующих языков: `english`,

`french`, `german` и т. п. Соответственно, если вы собираетесь писать текст на нескольких языках, надо подключить пакет `babel`, указав ему соответствующие опции. При этом «основной» язык, правила которого будут применяться по умолчанию, следует указать в этом списке последним. Например, запись в преамбуле

```
\usepackage[english,german,french]{babel}
```

означает, что основная часть текста будет написана по-французски с фрагментами на немецком и английском. Переход к фрагменту текста, написанному на другом языке, можно осуществить по меньшей тремя способами. Во-первых, можно установить другой основной язык с помощью команды `\selectlanguage`, единственный аргумент которой — название требуемого языка (читай: стилевой опции пакета `babel`):

```
\selectlanguage{english}
```

Здесь и далее в качестве «нового языка» можно указывать только то, что было указано в качестве опции пакета `babel`, никакие языки сверх этого таким способом добавить не получится. Второй способ, удобный для включения небольшого фрагмента на другом языке, — воспользоваться командой `\foreignlanguage` с двумя аргументами: первый аргумент — название языка, второй аргумент — текст. Пример:

```
\foreignlanguage{english}{What's up?}
```

(если основной язык — французский, так имеет смысл сделать, чтобы вопросительный знак в английской фразе напечатался без отступа от предыдущего слова). Наконец, для включения более обширного фрагмента можно воспользоваться окружением `otherlanguage` с единственным аргументом — названием языка:

```
\begin{otherlanguage}{english}
...
\end{otherlanguage}
```

Для каждого известного пакету `babel` языка могут быть определены команды и сочетания символов, призванные воспроизвести специфические для этого языка полиграфические эффекты. Например, в пакете `babel` с опцией `french` определена команда `\er`, с помощью которой, написав в `tex`-файле `1\er`, на печати можно получить 1<sup>er</sup> («первый» в мужском роде). Для каких-то языков таких дополнительных команд и сочетаний символов может быть очень много, для каких-то — совсем мало: это уже зависит от того, насколько востребован  $\text{\TeX}$  в той или иной стране и насколько активны тамошние  $\text{\TeX}$ ники. Все подробности можно узнать в документации к пакету, которую лучше всего взять с сайта CTAN'а (см. начало гл. VIII).

## И.5. Практические выводы для пишущих по-русски

Подытожим в заключение (в форме рецептов без пояснений), как лучше всего оформлять ЛАТЭХ'овский файл с русским текстом.

**Первый способ (только для использующих «виндовую» кодировку; с недостатками).** Написать в преамбуле файла

```
\usepackage[T2A]{fontenc}
\usepackage[russian]{babel}
```

После этого все русские буквы, кроме ё и Ё, набирать непосредственно, ё и Ё набирать как `\`е` и `\`Е`. Смириться с тем, что в словах с набранными так ё и Ё автоматический перенос будет невозможен. Смириться с тем, что при использовании пакета `hyperref` русские заголовки в закладках pdf-файла, полученного из `tex`-файла, будут отображаться некорректно (см. приложение Г).

**Второй способ (для всех восьмибитных кодировок файлов; с теми же недостатками).** Написать в преамбуле файла

```
\usepackage[T2A]{fontenc}
\usepackage[russian]{babel}
```

но обрабатывать файл не просто так, а с помощью следующей командной строки:

```
latex --translate-file имя_tcx-файла.tcx имя_файла
```

(и аналогично для `pdflatex`), где *имя\_tcx-файла* — `win2t2`, если используется «виндовая» кодировка `cp1251`, `alt2t2`, если используется «досовская», или «альтернативная», кодировка `cp866`, и `koi2t2`, если используется «юникопская» кодировка `koi8-r`.

Смириться с тем, что при использовании пакета `hyperref` русские заголовки в закладках pdf-файла, полученного из `tex`-файла, будут отображаться некорректно (см. приложение Г).

Соответствующие `tcx`-файлы надо либо найти в интернете (на момент написания этих строк это было легко), либо написать самому, либо, наконец, попросить об этом вашего системного администратора (простая инструкция приведена выше в разд. И.2). Далее `tcx`-файлы надо поместить в место, где ТЭХ будет их «видеть», плюс, возможно, надо будет объяснить ТЭХ'у, что набор видимых ему файлов изменился. Некоторые подробности см. в приложении О.



**Третий способ (универсальный; с другими недостатками).** Написать в преамбуле файла

```
\usepackage[кодировка]{inputenc}  
\usepackage[russian]{babel}
```

(где *кодировка* — cp1251, cp866, koi8-r или utf8). Обрабатывать файл обычными командами `latex` или `pdflatex`, без ключей, указывающих на перекодировку. Порадоваться, что при использовании пакета `hyperref` русские заголовки в закладках pdf-файла будут отображаться корректно, по крайней мере при использовании любой из восьмибитных кодировок в `tex`-файле. Смириться с тем, что `toc`-файл будет нечитаемым, а `idx`-файл к тому же будет невозможно обработать программой `makeindex`.

## Приложение К. 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 — почтовый адрес автора (для обычной, не электронной почты). Этот адрес 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`. Электронный адрес нужно указать в команде `\email`. Команда `\address` должна следовать *после* команды `\author`; команды `\curraddr` (если есть) и `\email` 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 `\curraddr` (если нужно) и `\email`.

Если вы хотите посвятить кому-то свою работу, запишите это посвящение в аргументе команды `\dedicatory..` Список ключевых слов, определяющих вкратце, что надо знать для понимания вашего труда, следует указывать в аргументе команды `\keywords`; в аргументе команды `\subjclass` можно 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»  $\text{\LaTeX}$ 'у, состоит в командах рубрикации. Классы `amsart` и `amsproc` допускают в качестве таковых известные нам `\section`, `\subsection`, `\subsubsection`, а также еще и 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 М. Метапост

Метапост (MetaPost) — это поставляющаяся вместе с  $\text{\TeX}$ ’ом система, предназначенная для создания рисунков, которые удобно встраивать в  $\text{\LaTeX}$ ’овские документы с помощью пакета `graphicx`. Как  $\text{\TeX}$  не является текстовым редактором или системой типа WYSIWYG, так и Метапост не является графическим редактором — это язык программирования, предназначенный для описания рисунков. Он разработан Джоном Хобби (John Hobby) по образцу (и на основе) созданного Дональдом Кнудом языка METAFONT, предназначенного для описания шрифтов (см. приложение А). В этом приложении мы не пытаемся дать полное описание Метапоста (такое описание входит в комплект поставки  $\text{\TeX}$ ’а, а также легко находится в интернете по запросу «metapost manual»): мы всего лишь познакомим читателя с основными принципами в надежде, что он войдет во вкус и самостоятельно разберется в дальнейших подробностях.

### М.1. Общая картина. Рисунки без кривых. Надписи

Чтобы «нарисовать» рисунок в Метапосте, нужно с помощью текстового редактора создать его описание. Это описание надо сохранить в файл с расширением `.mp` (в одном `mp`-файле могут содержаться описания нескольких рисунков). Затем надо оттранслировать этот файл (соответствующая команда обычно называется `mpost` или `mp`; если ваш `mp`-файл называется, скажем, `mypics.mp`, то в командной строке надо будет сказать `mpost mypics`). В результате получится один или несколько файлов с рисунками (в формате, близком к Encapsulated PostScript), которые можно уже вставлять в  $\text{\LaTeX}$ ’овский документ.

Рассмотрим теперь конкретный пример рисунка, сделанного в Метапосте, — треугольник с тремя высотами (рис. М.1).

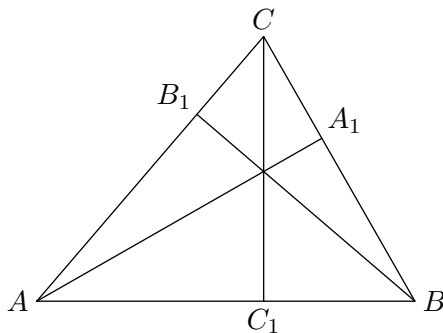


Рис. М.1. Треугольник  $ABC$  с проведенными высотами

Он был получен из следующего `mp`-файла.

```

beginfig(1)

u:=1cm;
pair a, b, c, aa, bb, cc;
numeric t[];

a:=(0,0);
b:=(5u,0);
c:=(3u,3.5u);

aa=t1[b,c]; (b-c) dotprod (a-aa)=0;
bb=t2[a,c]; (a-c) dotprod (b-bb)=0;
cc=t3[a,b]; (a-b) dotprod (c-cc)=0;

draw a--b--c--a;
draw a--aa; draw b--bb; draw c--cc;

label.lft(btex  $A$  etex, a);
label.rt(btex  $B$  etex, b);
label.top(btex  $C$  etex, c);
label.urc(btex  $A_1$  etex, aa);
label.ulft(btex  $B_1$  etex, bb);
label.bot(btex  $C_1$  etex, cc);

endfig;
end;

```

Разберем, как он устроен.

Всякий `mp`-файл заканчивается командой `end`; описание каждого рисунка начинается строкой `beginfig` с номером в круглых скобках, а заканчивается командой `endfig`; номера различных рисунков не должны совпадать. Если, например, ваш файл называется `mypics.mp`, то рисунок, описываемый в нем между строками `beginfig(183)` и `endfig`, после трансляции запишется в виде файла `mypics.183`<sup>1</sup>.

Что же, однако, написано между `beginfig` и `endfig`? Первая содержательная строка имеет вид `u:=1cm`; — в ней мы присваиваем переменной `u` значение в один сантиметр; далее мы будем использовать эту переменную как единицу длины (можно было бы этой переменной вообще не

---

<sup>1</sup>Если будете пробовать самостоятельно построить рисунок из примера, не забудьте, что для обработки графического файла `pdflatex`'ом его надо переименовать, чтобы расширение было не `.1`, а `.mps` — см. разд. IV.9.1.

вводить, а задавать все расстояния явно, но это менее удобно: если рисунок понадобится немного пропорционально уменьшить или увеличить, то будет достаточно изменить в файле только эту строку, присвоив `u` другое значение). В следующих двух строках идет объявление переменных. Шесть переменных от `a` до `ss` имеют тип `pair`. Переменные такого типа суть пары чисел, обычно они используются для задания точек на плоскости или векторов (ниже мы увидим примеры использования в обоих смыслах). В переменных `a`, `b` и `c` будут записаны координаты вершин нашего треугольника, а в переменных `aa`, `bb` и `ss` — координаты точек  $A_1$ ,  $B_1$  и  $C_1$ .

В строке

```
numeric t[];
```

объявляется массив переменных типа `numeric` (число). Как и во многих других языках, к элементу такого массива можно обратиться как к `t[i]`, где `i` — число или переменная типа `numeric`; если, однако, номер элемента в массиве — явно заданное натуральное число, то квадратные скобки можно опустить (ниже мы обращаемся к трем элементам этого массива как к `t1`, `t2` и `t3`).

Внимательный читатель спросит, почему же мы тогда не описывали переменную `u`. Ответ: все неописанные переменные считаются имеющими тип `numeric`. Впрочем, советуем этим свойством Метапоста не злоупотреблять.

Далее мы присваиваем значения переменным `a`, `b` и `c`, то есть задаем координаты вершин треугольника. Обратите внимание, что нули мы написали «просто так», без `u`. С точки зрения Метапоста всякая переменная типа `numeric` (в том числе любая из компонент переменной типа `pair`) — это просто число. Если не указана единица измерения, то она трактуется как соответствующее количество пунктов: если бы мы хотели присвоить переменной `u` значение 30 пунктов, то достаточно было бы написать `u:=30;`.

Отметим еще, что в Метапосте используются не  $\TeX$ 'овские пункты, равные  $1/72,27$  дюйма, а «постскриптовские», равные в точности  $1/72$  дюйма.

Далее начинается самое интересное: мы находим основания высот средствами самого Метапоста! Разберемся с точкой  $A_1$ , координаты которой будут записаны в переменной `aa`. В строке

$$aa=t1[b,c]; (b-c) \text{ dotprod } (a-aa)=0; \quad (1)$$

записаны сразу две метапостовские команды (можно было бы записать их и в две строки, это неважно), в которых используется знак равенства

без двоеточия; дело в том, что это не присваивания, а уравнения. В правой части первого уравнения используется следующая метапостовская конструкция: если написать

$$t[b, c],$$

где  $b$  и  $c$  — переменные типа `pair` (или вообще любые выражения со значением типа `pair`), то значением этого выражения будет  $b + t(c - b)$  (так представляется любая точка, лежащая на прямой  $bc$ ). Поскольку переменной `t1`, она же `t[1]`, мы никакого значения пока не присваивали, уравнение `aa=t1[b,c]`; означает буквально следующее: «точка `aa` лежит где-то на прямой  $bc$ ». Во втором из уравнений используется операция `dotprod`; это операция на двух переменных типа `pair`, результат которой — скалярное произведение соответствующих векторов<sup>2</sup>. Поскольку скалярное произведение двух векторов равно нулю тогда и только тогда, когда они перпендикулярны, второе уравнение в (1) означает, что прямая, соединяющая  $a$  и  $aa$ , перпендикулярна прямой, соединяющей  $b$  и  $c$ . В совокупности с условием « $aa$  лежит на прямой  $bc$ » из первого уравнения это условие точку  $aa$  полностью определяет и означает, что она является основанием высоты (заодно мы полностью определили и значение переменной `t1`, хотя оно нам и не нужно). Итак, Метапост сам решил систему из трех уравнений с тремя неизвестными (три — это две координаты точки плюс параметр `t1`). В общем случае Метапост может решить любую систему *линейных* уравнений, при условии, что ее решение существует и единственно. Уравнения не обязательно записывать подряд: при появлении каждого нового уравнения программа включает его в систему вместе с предыдущими.

Аналогично (в двух следующих строках) находятся координаты будущих точек  $B_1$  и  $C_1$ .

Для двух следующих точек мы использовали другие переменные `t2` и `t3` — если бы мы попробовали использовать для них ту же переменную, что и для  $A_1$ , то Метапост зафиксировал бы ошибку типа «несовместная система уравнений» (`inconsistent equation`): если значение `t1` из уже имеющихся уравнений однозначно определено, то его нельзя использовать в качестве неизвестной в следующем уравнении.

Итак, координаты всех нужных нам точек найдены, можно начинать рисовать. Это делается в следующих двух строчках кода. Как видите, для рисования используется команда `draw`, а для указания на то, что рисовать надо именно прямую, между двумя выражениями со значением типа `pair` (не обязательно именно переменными, как в нашем простом примере) следует поставить `--`.

---

<sup>2</sup>На всякий случай напомним, что скалярное произведение векторов с координатами  $(x_1, y_1)$  и  $(x_2, y_2)$  — число  $x_1y_1 + x_2y_2$ .

Осталось объяснить, как нанести на рисунок надписи. Как видно из нашего примера, это делается с помощью команды `label`. Объясним синтаксис. Если пока игнорировать то, что написано между словом `label` и открывающей скобкой, то в скобках через запятую указаны две вещи: сначала текст, который нужно надписать (обычно между `btex` и `etex`), а затем координаты точки (т. е., как водится, выражение со значением типа `pair`), в которую надо поместить надпись. Смысл же всех этих `.lft`, `.urt` и пр. между `label` и скобкой в том, что они задают расположение надписи относительно указанной в скобках точки. Для обладающих минимальными познаниями в английском смысл этих обозначений должен быть достаточно ясен: `.top` — надпись над точкой, `.bot` — надпись под точкой, `.lft` — надпись слева от точки, `.urt` — надпись сверху-справа от точки и т.п. — полный список см. в руководстве. Команду `label` можно использовать и без суффиксов наподобие `.bot` или `.urt` — в этом случае точка, заданная во втором аргументе, будет находиться в центре надписи (точнее, соответствующего ей T<sub>E</sub>X'овского «блока» — см. гл. VII).

Выше мы умолчали об одной важной подробности. Если вы используете вставку надписей с помощью `btex` и `etex`, то в дополнение к описанному выше надо сделать еще три вещи:

- 1) в начало `mp`-файла (до первого `beginfig`) вписать текст

```
verbatimtex
преамбула вашего LATEX'овского документа
etex
```

(преамбула должна включать и `\begin{document}`);

- 2) в конец `mp`-файла вписать текст

```
verbatimtex
\end{document}
etex
```

(после последнего `endfig`);

- 3) установить значение переменной среды T<sub>E</sub>X в `latex` (под Windows это делается через меню «панель управления» — «система» — «дополнительные параметры», под UNIX-подобными системами надо сказать что-нибудь вроде `export TEX=latex`).

В нашем примере без всего этого удалось обойтись, так как надписи были настолько простыми, что система более или менее корректно работала без дополнительных установок.

В разобранным нами примере самым неизящным было, пожалуй, объявление массива `t[]`, значения переменных в котором используются только один раз и неявно. В Метапосте есть способ без этого обойтись. Именно, волшебное слово `whatever` предназначено специально для



обозначения числовой переменной, явное значение которой нас не интересует. Опытный пользователь Метапоста вообще не стал бы объявлять массив `t[]`, а три строки с уравнениями записал бы следующим образом:

```
aa=whatever[b,c]; (b-c) dotprod (a-aa)=0;
bb=whatever[a,c]; (a-c) dotprod (b-bb)=0;
cc=whatever[a,b]; (a-b) dotprod (c-cc)=0;
```

Таким образом, преимущество **whatever** в том, что его не надо объявлять и можно использовать повторно, даже если значение подразумеваемого в этом слове числа уже зафиксировано из предыдущего уравнения.

На нашем рисунке все линии имеют одинаковую толщину. На самом деле Метапост позволяет рисовать разные линии по-разному: потолще, потоньше, пунктиром и т.п. Чтобы узнать, как добиться этих эффектов, поищите в руководстве по Метапосту ключевые слова **pen**, **withpen** и **dashed**; заодно можно ознакомиться и с командой **drawdot**, предназначенной для рисования жирных (или не очень) точек.

## М.2. Рисование кривых

Метапост позволяет рисовать не только прямые, но и гладкие кривые более или менее произвольной формы. Если, например,  $a$ ,  $b$ ,  $c$  и  $d$  — выражения со значением типа **pair** — переменные, объявленные как **pair**, или явно указанные пары чисел (в скобках, через запятую — см. пример из предыдущего параграфа), или более сложные выражения со значением типа **pair** — то команда

```
draw a..b..c..d;
```

нарисует плавную кривую, проходящую через точки  $a$ ,  $b$ ,  $c$  и  $d$  (если ее форма вас не устраивает, ничто не мешает задать больше промежуточных точек).

Кривые, которые рисует Метапост, составлены из так называемых кубических сплайнов Безье (они же кубические кривые Безье). Если, например, в **mp**-файле написано **draw a..b..c**, то каждый из участков от  $a$  до  $b$  и от  $b$  до  $c$  строится следующим образом. В дополнение к начальной точке  $p_1 = a$  и конечной точке  $p_4 = b$  Метапост выбирает две «управляющие точки» (control points)  $p_2$  и  $p_3$ . Далее построение сплайна можно описать двумя способами: геометрически и алгебраически. Геометрическое описание таково (рис. М.2). Сначала надо соединить отрезками точки  $p_{12}$ ,  $p_{23}$  и  $p_{34}$  — середины отрезков  $\overline{p_1p_2}$ ,  $\overline{p_2p_3}$  и  $\overline{p_3p_4}$ , затем соединить отрезком точки  $p_{123}$  и  $p_{234}$  — середины отрезков  $\overline{p_{12}p_{23}}$  и  $\overline{p_{23}p_{34}}$  и взять наконец точку  $p_{1234}$  — середину отрезка  $\overline{p_{123}p_{234}}$ . Точка  $p_{1234}$  — первая построенная точка нашей кривой. Теперь итерируем эту конструкцию: построим таким же способом точки на кривой, соответствующие четверкам точек  $p_1, p_{12}, p_{123}, p_{1234}$  и  $p_{1234}, p_{234}, p_{34}, p_4$ , и т.д. Этот процесс быстро сходится к искомой кривой.

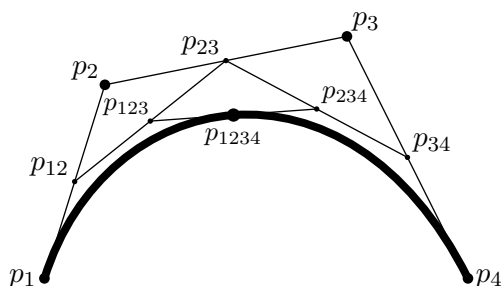


Рис. М.2. Кубический сплайн Безье

Ту же кривую очень просто описать алгебраически: кубический сплайн Безье с управляющими точками  $p_2$  и  $p_3$ , идущий из  $p_1$  в  $p_4$ , параметрически задается формулой

$$t \mapsto (1-t)^3 p_1 + 3(1-t)^2 t p_2 + 3(1-t)t^2 p_3 + t^3 p_4,$$

где  $t \in [0; 1]$ . И из геометрического, и из алгебраического описания кривой явствует, что прямая  $p_1 p_2$  — касательная к ней в точке  $p_1$ , а прямая  $p_3 p_4$  — касательная к ней в точке  $p_4$ .

Как видите, в описанном алгоритме существует большая свобода для выбора двух управляющих точек  $p_2$  и  $p_3$ . По умолчанию они выбираются на каждом из участков так, чтобы на кривой не было изломов и она имела «наиболее простую» форму; на этот выбор можно влиять разными способами (см. ниже); если вам зачем-то надо задать управляющие точки в явном виде, можно написать

```
draw a .. controls b and c .. d;
```

для кривой Безье, идущей из  $a$  в  $d$  с управляющими точками  $b$  и  $c$ .

Если между какими-то парами точек написать `--` вместо `..`, то соответствующий участок кривой будет прямолинейным. Если вместо `..` написать не `--`, а `---`, то криволинейные участки будут примыкать к прямолинейным без изломов.

Существует и ряд других способов повлиять на выбор формы кривой, проходящей через заданные вами точки. Расскажем еще об одном из них (прочее ищите в руководстве по ключевому слову `curve`). Именно, в каждой точке, через которую вы проводите кривую, можно в явном виде указать направление касательной к участку кривой, выходящей из этой точки. Например, команда

```
draw (0,0)..(3cm,0);
```

нарисует просто горизонтальный отрезок, а вот команда

```
draw (0,0){(-1,1)}..(3cm,0);
```

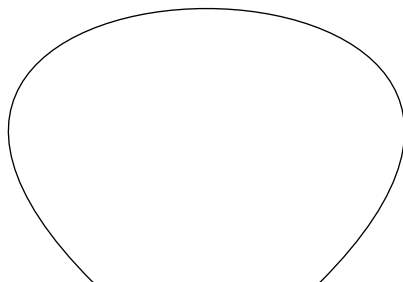


Рис. М.3. `draw (0,0){(-1,1)}..(5cm,0);`

нарисует кривую, касательная к которой в начале координат образует угол  $135^\circ$  с положительным направлением оси абсцисс (рис. М.3).

Общее правило: после точки можно указать в фигурных скобках выражение со значением типа `pair` (ненулевым, разумеется) — и тогда касательный вектор к участку кривой, выходящей из этой точки, будет сонаправлен с вектором, указанным в фигурных скобках. Длина этого вектора значения не имеет.

Для указания направления касательной к кривой часто бывает удобно пользоваться оператором `dir`. Именно, если  $t$  — переменная (или выражение) типа `numeric`, то `dir t` — это вектор длины 1, направленный под углом  $t$  градусов к положительному направлению оси абсцисс (направление отсчета — против часовой стрелки). В приведенном выше примере можно было вместо `{(-1,1)}` написать `{dir 135}`.

Разберем теперь более содержательный пример рисунка с кривой, с помощью которого мы проиллюстрируем еще несколько важных метапостовских конструкций.

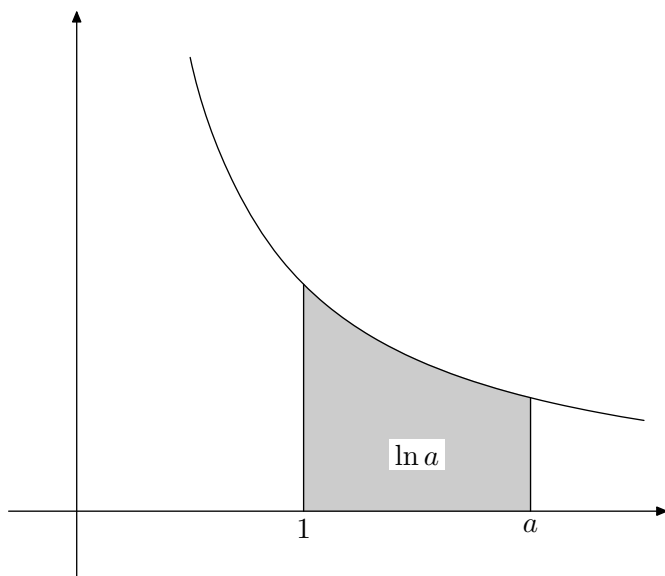
Итак, давайте посмотрим на рисунок М.4, иллюстрирующий геометрическое определение натурального логарифма.

Этот рисунок получен из следующего кода:

```
beginfig(2)

u:=3cm;
pair a, b, c, d;
path hyper, boundary;
picture ln;

hyper:=(0.5u,2u) for i=1 upto 10:
  ..((0.5+0.2*i)*u, (1/(0.5+0.2*i))*u)
endfor;
```

Рис. М.4.  $\ln a = \int_1^a \frac{dx}{x}$ .

```

a:=(u,0);
c:=(2u,0);
b:=hyper intersectionpoint (a--(u,infinity));
d:=hyper intersectionpoint (c--(2u,infinity));

boundary:=buildcycle(a--1.1[a,b], hyper, c--1.1[c,d], a--c);
filldraw boundary withcolor 0.8 white;

draw hyper;
draw a--b;
draw c--d;
drawarrow (-0.3u,0)--(2.6u,0);
drawarrow (0,-0.3u)--(0,2.2u);

label.bot(btex $1$ etex, a);
label.bot(btex $a$ etex, c);

ln:=thelabel(btex $\ln a$ etex, (1.5u,0.25u));
unfill bbox ln;
draw ln;

endfig;

```

Разберем этот код последовательно. После присваивания значения переменной `u`, которая будет нашей единицей длины, и объявления четырех переменных типа `pair` (это будут четыре вершины криволинейного четырехугольника на рисунке) идет объявление двух переменных типа `path` и одной переменной типа `picture`. Переменные типа `path` имеют своими значениями всевозможные кривые (включая отрезки и ломаные), а значения переменных типа `picture` — более или менее произвольные фрагменты рисунка. Переменная `hyper` будет обозначать участок графика гиперболы, а `boundary` — границу криволинейного четырехугольника. Смысл переменной `ln` мы объясним позднее.

В следующих трех строках мы задаем значение переменной `hyper`, то есть описываем эту кривую. Команды для рисования именно гипербол в Метапосте не предусмотрено, поэтому мы рисуем ее приближенно, как гладкую кривую, проходящую через 11 точек, лежащих на графике функции  $y = 1/x$ , с абсциссами, образующими арифметическую прогрессию от 0,5 до 2,5. Содержательно код

```
hyper:=(0.5u,2u) for i=1 upto 10:
  ..((0.5+0.2*i)*u, (1/(0.5+0.2*i))*u)
endfor;
```

равносильно следующему:

```
hyper:=(0.5u,2u) .. (0.7u, (1/0.7)*u) .. ⟨еще 8 точек⟩ .. (2.5u,0.4u);
```

вместо того, чтобы писать это длинное выражение вручную с риском ошибиться, мы воспользовались существующей в Метапосте конструкцией цикла в ее простейшей форме: переменная `i` последовательно принимает значения от 1 до 10 с шагом единица, и каждое из этих значений подставляется вместо `i` в последовательность символов

```
..((0.5+0.2*i)*u, (1/(0.5+0.2*i))*u)
```

— так что в результате «разворачивания» текста от `for` до `endfor` получается последовательность из десяти выражений вида  $..(x, y)$ , которая, вкупе с изначальным  $(0.5u, 2u)$ , подставляется в правую часть присваивания `hyper:=...;`.

Точка с запятой после `endfor` не является составной частью конструкции цикла: она необходима, так как необходимо заканчивать точкой с запятой любой оператор присваивания.

В Метапосте можно организовывать и более сложные циклы, о чем вы можете узнать из руководства.

В следующих четырех строках мы присваиваем значения переменным, обозначающим вершины криволинейного четырехугольника. Координаты нижних вершин `a` и `c` мы задаем непосредственно, а вот координаты верхних вершин `b` и `d` вычисляем специфичным для Метапоста

способом: мы задаем их как точки пересечения двух кривых<sup>3</sup>. В обоих случаях одна из кривых задана переменной **hyper**, а другая — просто вертикальный отрезок. Чтобы быть уверенными, что кривые действительно пересекутся, в качестве ординаты второго конца отрезка мы указали **infinity** — это число, задающее длину, много большую размеров любого разумного рисунка.

Если две кривые, точку пересечения которых мы хотим найти с помощью оператора **intersectionpoint**, не пересекаются, то получится сообщение об ошибке, а если они пересекаются в нескольких точках, то Метапост в качестве результата выдаст только одну из них, выбрав ее по одному ему ведомым соображениям.

На рис. М.4 криволинейный четырехугольник залит серым цветом; следующая строка кода задает границу заливаемой области. Объясним, как эта граница задается. С точки зрения Метапоста среди всех кривых выделяется их подкласс, называемый *циклами*. Геометрически цикл — это просто замкнутая кривая, но чтобы Метапост считал замкнутую кривую циклом, его надо об этом оповестить. Так, если **a**, **b** и **c** — три переменных типа **pair**, а **p** — переменная типа **path**, то после выполнения команды

```
p:=a--b--c--a;
```

значением переменной **p** будет замкнутая трехзвенная ломаная, но циклом она не будет; чтобы получился цикл, надо вместо последней точки, куда цикл должен вернуться, написать слово **cycle**:

```
p:=a--b--c--cycle;
```

В нашем случае тоже можно было бы задать искомый цикл с помощью ключевого слова **cycle**, написав что-нибудь вроде

```
boundary:=(u,0)--(u,u)
for i=0 upto 10:
  ..((1+0.1*i)*u,(1/(1+0.1*i))*u)
endfor
--(2u,0)--cycle;
```

но проще и поучительней воспользоваться другой метапостовской конструкцией, позволяющей построить цикл из уже готовых кривых: команда **buildcycle** постарается смастерить цикл из кривых, переданных ей в качестве аргументов (ваша забота — чтобы такой цикл существовал и был единственен). Мы написали **a--1.1[a,b]** вместо **a--b** (то есть

---

<sup>3</sup>В данном случае, так как кривая **hyper** — график функции, заданной явным уравнением, можно было бы указать координаты этих точек непосредственно, как **(u,u)** и **(2u,0.5u)**; в учебных целях мы воспользовались более общим методом.

немного продлили отрезок за точку *b*), чтобы гарантировать, что цикл действительно замкнется: пересечения кривых, не являющихся прямыми, Метапост находит не точно, а приближенно, так что из-за ошибки округления ему может показаться, что какие-то кривые не пересекаются.

В следующей строке, коль скоро цикл — граница заливаемой части уже определен, мы проводим собственно заливку.

Рисовать кривые и прямые рекомендуется уже после заливки; это мы и делаем в следующих пяти строчках кода. Обратите внимание на ранее не встречавшуюся команду `drawarrow`, предназначенную для рисования кривых и отрезков, заканчивающихся стрелкой. Внешний вид стрелки также можно регулировать.

Далее мы уже знакомым нам способом наносим на чертеж надписи *l* и *a*. После этого остается вырезать в заливке светлый прямоугольник подходящих размеров и написать в нем надпись `ln a`. Этому посвящена оставшаяся часть кода. Посмотрите сначала на строку, в которой присваивается значение переменной `ln`. В правой части стоит команда `thelabel`, аналогичная знакомой нам команде `label`; разница в том, что `thelabel` ничего не рисует — ее можно лишь использовать для получения выражений типа `picture`. У команды `thelabel` возможны те же суффиксы, что у `label`; так как в данном случае суффиксов нет, центр надписи лежит в точке `(1.5u, 0.25u)`.

Итак, «картинка» `ln` определена. В следующей строке мы расчищаем место для надписи: оператор `bbox`, примененный к выражению типа `picture`, дает кривую — границу наименьшего прямоугольника, в который вписана «картинка», причем эта кривая является циклом с точки зрения Метапоста. Команда же `unfill`, будучи примененной к кривой-циклу, стирает с рисунка все, что находится внутри цикла. После этого остается сказать `draw ln`; — и на расчищенном нами месте появится искомая надпись.

Мы упомянули далеко не о всех возможностях Метапоста. Например, можно обрезать рисунок по границе данного цикла, можно найти касательное направление (и соответственно нарисовать касательную) в произвольной точке кривой, можно подвергать рисунок в целом или любые его элементы различным преобразованиям (гомотетии, сдвигам, поворотам, симметриям, подобиям и вообще любым аффинным преобразованиям), рисунки можно делать разноцветными, можно для сокращения и упрощения текста определять собственные макросы — обо всем этом также можно узнать из руководства.

## Приложение О. Откуда взять Т<sub>Е</sub>X?

Как мы уже отмечали, все основные реализации Т<sub>Е</sub>X’а распространяются бесплатно. На данный момент такими реализациями являются MiK<sub>Т</sub>ЕX (под Windows), te<sub>Т</sub>ЕX (под UNIX-подобными системами) и Т<sub>Е</sub>Xlive (под UNIX-подобными системами и Mac OS). В интернете их можно найти много где, но надежнее всего — в архиве CTAN (Comprehensive Т<sub>Е</sub>X Archive Network), о котором мы уже неоднократно упоминали; на момент написания этих строк адрес CTAN’а был <http://www.ctan.org>.

В поставку Т<sub>Е</sub>X’а входят Bib<sub>Т</sub>ЕX и Метапост (они тоже распространяются бесплатно), а также все рекомендуемые нами в этой книге стилевые пакеты. Мы рекомендуем ставить сразу максимальную комплектацию. Если по той или иной причине нужного стилевого пакета (шрифта, вспомогательного файла и т. п.) в вашей установке нет, его надо взять с CTAN’а и разложить полученные файлы по тем директориям, в которых они должны находиться. Стилиевой файл можно поместить и в ту директорию, в которой вы сейчас работаете, но тогда Т<sub>Е</sub>X не сможет его «увидеть» из других директорий; если вы хотите воспользоваться добытым (или написанным) вами стилевым файлом более чем в одном проекте, его надо положить в директорию, которую Т<sub>Е</sub>X просматривает всегда (для ряда других нужных Т<sub>Е</sub>X’у файлов другой возможности и нет). Куда именно помещать файлы, зависит от реализации Т<sub>Е</sub>X’а, которой вы пользуетесь — тут надо ознакомиться с документацией. Например, на компьютере, на котором я пишу эти строки, установлен MiK<sub>Т</sub>ЕX (откуда можно заключить, что этот компьютер работает под Windows). Все, относящееся к Т<sub>Е</sub>X’у (включая исполняемые файлы), находится в директории D:\miktex24; в этой директории имеются многочисленные поддиректории. Скажем, все, относящееся к Bib<sub>Т</sub>ЕX’у, лежит в директории D:\miktex24\texmf\bibtex, в которой, в свою очередь, расположены поддиректории bst, csf и bib (кто ознакомился с приложением Б, тот поймет, что именно в них расположено; в готовых bib-файлах содержатся образцы записи библиографий). А стилевые и им подобные файлы, относящиеся к Л<sub>А</sub>T<sub>Е</sub>X’у (точнее говоря, файлы, которые могут подвергнуться \input’у в процессе трансляции Л<sub>А</sub>T<sub>Е</sub>X’овского файла), находятся в директории D:\miktex24\texmf\tex\latex и ее поддиректориях.

Однако же если в процессе работы над текстом у вас появился новый стилевой файл, то помещать его именно в эту директорию или ее поддиректории неразумно: для таких целей в Т<sub>Е</sub>X’овском дереве предусмотрена специальная поддиректория, обычно называемая localtexmf, и именно в нее и ее поддиректории лучше складывать самодельные стиле-



вые файлы, а также стилевые файлы, которые вы раздобыли с CTAN'a или еще откуда-то.

Обычно стилевые пакеты скачиваются с CTAN'a в виде директории (иногда с поддиректориями), в которой уже содержится все необходимое — тогда эту директорию надо просто переместить в директорию `localtexmf`, как объяснялось выше. Иногда, однако, оказывается, что собственно стилового пакета (файла с расширением `sty`) и нет. Например, вы скачали стилевой пакет `ghnm` (не знаю, есть ли такой), но файла `ghnm.sty` не обнаруживается. В этом случае должны найтись файлы `ghnm.dtx` и `ghnm.ins`; если сказать в командной строке

```
latex ghnms.ins
```

то будет порожден и искомый файл `ghnm.sty`. (А если обработать с помощью  $\LaTeX$ 'а файл с расширением `dtx`, то получится документация к стиловому пакету.)

Два заключительных замечания. Во-первых, если имя подключаемого вами файла, лежащего в текущей директории, совпадает с именем какого-то из уже имеющихся стилевых файлов, то подключен будет именно файл из текущей директории, так что опасаться такого совпадения не надо. Во-вторых, при помещении нового файла в директорию, просматриваемую  $\TeX$ 'ом (чтобы в дальнейшем этот файл был виден отовсюду) одного копирования файла в нужную директорию недостаточно: современные реализации  $\TeX$ 'а не просматривают всякий раз дерево директорий заново, а поддерживают внутреннюю базу данных, из которой по имени файла можно быстро выяснить, где его искать. Соответственно, после добавления нового файла к  $\TeX$ 'овскому дереву необходимо эту базу данных обновить. В  $\text{MiK}\TeX$ 'е для этого надо в командной строке сказать `initexmf -u`.

## Приложение П. Презентации в Л<sup>A</sup>T<sub>E</sub>X'e

Для подготовки презентации в Л<sup>A</sup>T<sub>E</sub>X'e надо использовать класс `beamer`. Возможности его очень широки (последняя версия руководства содержит более двухсот страниц, и именно к ней может обращаться заинтересованный читатель). Полезно также посмотреть файлы с примерами презентаций, которые прилагаются к документации. В любом случае мы ограничимся краткой инструкцией.

Разумеется, исходный файл презентации должен начинаться строкой

```
\documentclass{beamer}
```

Как всегда в Л<sup>A</sup>T<sub>E</sub>X'e, после этого надо подключить все необходимые пакеты (например, `babel`, `amsmath` и другие), после чего (все еще в преамбуле) приступить к установке параметров самого `beamer`'а. Заметим, что `beamer` предоставляет большое количество всевозможных «тем оформления» (стилей оформления презентации), и читатель может выбрать таковую по своему вкусу. Мы ограничимся рекомендацией включить в преамбулу строку

```
\usetheme{Warsaw}
```

(почему-то большинство тем названы именами различных населенных пунктов). Далее в преамбуле следует указать название доклада и сведения о его авторах. Они указываются с помощью стандартных для Л<sup>A</sup>T<sub>E</sub>X'a команд `\title`, `\author` и `\date`, допускающих необязательные аргументы (например, в обязательный аргумент команды `\title` следует включить полное название доклада, а в необязательный — краткое, которое идет в колонтитул). Отметим также команду `\institute` (также допускающую необязательный аргумент), в которой указываются организации, с которыми аффилированы авторы. Вот пример:

```
\author[Иванов, Петров]{  
  И.\,И.\,Иванов\inst{1,2}  
  \and\underline{П.\,П.\,Петров\inst{2}}  
\institute{  
  \inst{1}Школа чародейства и волшебства  
  \and\inst{2}НИИ ЧаВо}
```

Здесь все понятно: организации нумеруются с помощью команды `\inst` внутри обязательного аргумента команды `\institute`, а ссылки на них ставятся внутри обязательного аргумента команды `\author`. Важно заметить, что номера организаций ставятся вручную.

На этом преамбула завершается. Каждый слайд презентации представляет собой окружение `frame` с одним обязательным аргументом (название слайда), внутрь которого помещается непосредственно содержимое слайда. В частности, в самом начале презентации рекомендуем вставить два следующих слайда:

```
\begin{frame}{}
  \maketitle
\end{frame}
\begin{frame}{Содержание}
  \tableofcontents
\end{frame}
```

Первый слайд, как легко понять, будет титульным, а на втором слайде будет приведено оглавление. Отметим важный момент: можно разделить презентацию на разделы и подразделы (с помощью команд `\section`, `\subsection` и `\subsubsection`). Названия разделов идут в оглавление и колонтитулы (в зависимости от используемой темы), а названия слайдов отображаются только на самом слайде.

Если хочется какой-то кусок текста выделить в красивую рамочку, можно воспользоваться окружением `block`:

```
\begin{block}{Великая теорема Ферма}
При  $n \geq 3$  уравнение  $x^n + y^n = z^n$ 
не имеет решений в натуральных числах
 $x$ ,  $y$  и  $z$ .
\end{block}
```

В обязательном аргументе стоит заголовок к тексту в рамочке, но его можно оставить пустым.

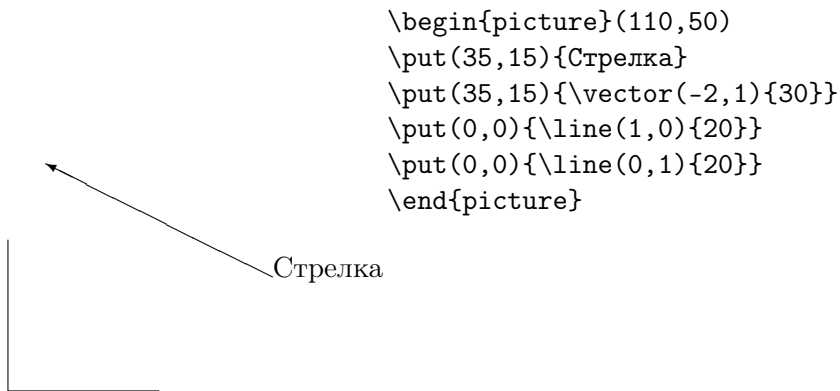
Если хочется, чтобы при показе презентации текст показывался не сразу, а поэтапно, полезно воспользоваться в соответствующих местах командой `\pause`. Также рекомендуем обратить внимание на команду `\only` (подробности — в документации к классу `beamer`).

Наконец, отметим, что транслировать презентацию лучше сразу `pdflatex`’ом: обработать ее обычным `latex`’ом тоже можно, но `dvi`-файл посмотреть вряд ли получится.

## Приложение Р. Рисунки с помощью подручных средств

Создатель  $\text{\LaTeX}$ 'а Лесли Лэмпорт предусмотрел возможность создания примитивных рисунков, состоящих из прямых и наклонных (с ограниченным репертуаром наклонов) линий, стрелок и окружностей (наклонные линии, стрелки и окружности собираются из символов специальных шрифтов, разработанных Лэмпортом именно для этих целей). Эта довольно-таки убогая технология может все же пригодиться в простых случаях.

Псевдорисунки создаются с помощью окружения `picture`. Разберем пример.



Во-первых, после `\begin{picture}` надо указать размер псевдорисунка. Эти размеры задаются в *круглых* скобках через запятую немедленно после `\begin{picture}`, сначала ширина, затем высота. Между скобками, запятой и числами, задающими размеры псевдорисунка, не должно быть пробелов (конец строки — тоже пробел). По умолчанию все размеры, относящиеся к псевдорисункам, задаются в пунктах (как в нашем примере). Можно указать любую единицу измерения размеров, относящихся к псевдорисункам: для этого надо изменить значение параметра `\unitlength` (см. с. 18 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, размер псевдорисунка задан. Для собственно «рисования» 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; для остальных трех команд, размещавших на рисунке стрелку и два отрезка, в фигурных скобках помещается описание стрелки и отрезков.

Когда мы говорили о координатах объекта, имелись в виду координаты так называемой «точки отсчета» на этом объекте. Точка отсчета стрелки — ее начало. Если объект — текст, то точка отсчета — это, грубо говоря, его левый нижний угол. Точнее говоря, объект рассматривается как «блок» (см. гл. VII), и его точка отсчета — просто точка отсчета блока.

Внутри окружения `picture` не должно быть пустых строк. Весь псевдорисунок, порожаемый этим окружением, рассматривается  $\text{\TeX}$ ’ом как одна большая буква (точнее говоря, как блок — см. гл. VII), ширина и высота которой заданы после `\begin{picture}` в скобках через запятую, так что естественное место таких «картинок» — внутри окружения `figure`.

Отрезки задаются с помощью команды `\line`.  $\text{\LaTeX}$ ’у надо сообщить наклон и размер отрезка. 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 примере выше имеются отрезки наклона  $(1,0)$  (горизонтальный) и  $(0,1)$  (вертикальный). 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.

Размер отрезка задается в фигурных скобках после круглых скобок, в которых задан наклон. Этот «размер» — длина его проекции на горизонтальную ось (или на вертикальную, если отрезок вертикален). Размер отрезка, в отличие от наклона, можно задавать произвольно (если он не слишком мал).

Стрелки задаются с помощью команды `\vector` с тем же синтаксисом, что и у команды `\line`. Отличие только в том, что для стрелок репертуар возможных наклонов еще более скуден, чем для отрезков: целые числа, задающие наклон, не должны превосходить 4 по абсолютной величине (и по-прежнему не должны иметь общих делителей). Как мы уже отмечали, точкой отсчета стрелки является ее начало.

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. Например, команда

```
\put(70,50){\circle{30}}
```

(внутри окружения `picture`) помещает на картинку круг диаметром 30, причем центр его оказывается в точке с координатами (70, 50) (единицей длины, как обычно, является значение параметра `\unitlength`, а если это значение не установлено, то пункт).

Количество реально возможных диаметров кругов ограничено. Если окружности или круга с диаметром, указанным в качестве аргумента команды `\circle` или `\circle*`, в  $\text{\LaTeX}$ ’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. Например, овал размером  $100 \times 80$  с центром в точке (50; 40) задается командой

```
\put(50,40){\oval(100,80)}
```

Возможны и «неполные» овалы, представляющие собой половины или четверти от полных. Чтобы задать неполный овал, надо задать команде `\oval` необязательный аргумент (в квадратных скобках, после обязательного). Для половины овала этот аргумент должен быть одной из букв `t` (верхняя половина), `b` (нижняя половина), `r` (правая половина) или `l` (левая половина). Для четверти надо указать в необязательном аргументе подходящее сочетание двух из этих букв (например, `tr` для правой верхней четверти).

Наконец, внутри окружения `picture` можно рисовать и кривые более или менее произвольной формы (так называемые квадратичные кривые Безье), составляемые из сотен черных квадратиков. Для этого используется команда `\qbezier` (внимание: *без* команды `\put`!). Пример задания кривой, заодно иллюстрирующий синтаксис:

```
\qbezier(22,2)(120,20)(20,77)
```

Эта запись означает, что кривая идет из точки с координатами (22; 2) в точку с координатами (20; 77), причем «на выходе» ее касательная

направлена от точки  $(22; 2)$  к точке  $(120; 20)$ , а «на входе» касательная направлена от точки  $(120, 20)$  к точке  $(20; 77)$ .

Если вы дошли до того, что пользуетесь в окружении **picture** командой **\qbezier**, настоятельно рекомендуем перейти на более совершенную систему создания рисунков (хотя бы на тот же Метапост): кривые при этом будут смотреться лучше, не говоря уж о многом другом.

## Приложение Ц. Цвет в L<sup>A</sup>T<sub>E</sub>X'e

Получаемые с помощью L<sup>A</sup>T<sub>E</sub>X'a pdf-файлы можно сделать цветными. Для этого используется стилевой пакет `color`.

Если, например, мы хотим, чтоб в словосочетании «Красная шапочка» первое слово было набрано красными буквами, то надо (подключив, разумеется, пакет `color`) написать так:

```
\textcolor{red}{Красная} шапочка
```

Команда `\textcolor` аналогична командам наподобие `\textbf`, но она принимает два аргумента: первый — название цвета, а второй — текст, который вы хотите в этот цвет выкрасить.

В качестве названия цвета заведомо можно использовать `black`, `white`, `red`, `green`, `blue`, `cyan` (цвет морской волны), `magenta` (малиновый) или `yellow`. Можно также определить свой цвет (см. ниже).

Наряду с командой `\textcolor` в пакете `color` определена команда `\color`, относящаяся к `\textcolor` так же, как `\bfseries` относится к `\textbf`: она меняет цвет текущего шрифта на указанный в ее (единственном) аргументе, , и если эта команда дана внутри группы, то по выходе из группы ее действие прекращается:

```
{\color{red}Красная} Шапочка
```

Можно также раскрасить фон в целом блоке; это делается с помощью команды `\colorbox`, аналогичной по своему действию команде `\fbox`, но принимающей дополнительный аргумент — название цвета. Вот как, для той же красной шапочки, написать слово «Красная» на красном фоне (полиграфист сказал бы: «на красной плашке»).

```
\colorbox{red}{Красная} Шапочка
```

Если вы хотите задать цвет, отличный от перечисленных выше, надо воспользоваться командой `\textcolor` (соответственно, `\color`, `\colorbox`) с необязательным аргументом, славящимся перед обязательными. В необязательном аргументе указывается «цветовая модель» — `rgb`, `cmuk` или `gray` (последняя — для оттенков серого цвета), а в первом обязательном аргументе, где мы ставили словесное обозначение цвета, надо тогда поставить его цифровое обозначение: одно число из отрезка  $[0; 1]$  для модели `gray` (насыщенность серого), три таких числа через запятую для модели `rgb` и четыре таких числа через запятую для модели `cmuk` (поскольку числа перечисляются через запятую, в качестве разделителя для дробной части в них надо использовать десятичную точку). Примеры:



```
\textcolor[gray]{0.75}{Серая} Шейка  
\textcolor[rgb]{0.1,0.2,0.68}{Красная} Шапочка
```

Если определяемый «вручную» цвет встречается в вашем файле часто, можно определить для него отдельное имя с помощью команды `\definecolor` с тремя аргументами (все обязательные!): придуманное вами название цвета, затем модель, затем цифровое обозначение:

```
\definecolor{serburmal}{rgb}{0.1,0.2,0.68}
```

После такого определения название цвета `serburmal` можно будет использовать наравне с `red` или `blue`.

Сведения об не рассмотренных нами возможностях пакета `color` можно, как водится, узнать из документации. Отметим еще, что все сказанное относилось исключительно к созданию цветных pdf-файлов для обмена с коллегами или размещения в сети; подготовка макетов для цветной печати в типографии — дело более сложное, и этих вопросов мы не касаемся.

## Приложение Ч. Что читать дальше

Для читателей, решивших изучить  $\TeX$  поглубже, приведем рекомендации по дальнейшему чтению.

Начнем с того, что разумный ответ на практически любой осмысленный вопрос по  $\TeX$ ’у можно быстро получить на (англоязычном) форуме [tex.stackexchange.com](http://tex.stackexchange.com); за годы работы этого сайта в его архиве скопилось множество полезных рекомендаций, в том числе и рассчитанных на начинающих.

Далее, 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 VIII. 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  $\TeX$ book [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 находятся в свободном доступе, 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`- или `pdf`-файл, в котором обычно записано руководство по использованию пакета плюс исходный текст пакета с комментариями. В приложении О мы объясняли, как обработать `dtx`-файл с помощью  $\LaTeX$ ’а, чтобы получить из него стилевой пакет. Для подключения шрифтов, отличных от стандартных,  $\LaTeX$  использует так называемые `fd`-файлы; для них роль `dtx`-файлов играют `fdd`-файлы, обращаться с которыми надо так же, как с `dtx`-файлами.

В некоторые установки  $\TeX$ 'а  $\text{dtx}$ -файлы не включаются с целью экономии места, но тогда обычно поставляются сгенерированные из них  $\text{dvi}$ -файлы. В любом случае все  $\text{dtx}$ - и  $\text{fdd}$ -файлы есть на CTAN'е (см. приложение О).

Все компоненты ядра  $\text{\LaTeX}$ 'а также доступны в  $\text{dtx}$ -формате и открыты для изучения. Остается пожелать читателю успеха в этом непростом деле.

## Литература

- [1] L. Lamport. *L<sup>A</sup>T<sub>E</sub>X. A Document Preparation System, User's Guide and Reference Manual*. — Addison-Wesley, 1994. (Первое издание вышло в 1985 г; в нём описана система L<sup>A</sup>T<sub>E</sub>X 2.09.)
- [2] D. E. Knuth. *The T<sub>E</sub>Xbook*. Addison-Wesley, 1984. Русский перевод: Дональд Э. Кнут. *Все про T<sub>E</sub>X*. — Протвино: РДТ<sub>E</sub>X, 1993.
- [3] H. Partl, E. Schlegl, I. Hyna. *L<sup>A</sup>T<sub>E</sub>X-Kurzbeschreibung*. — Пособие в электронном виде; входило в состав дистрибутива emT<sub>E</sub>X — реализации T<sub>E</sub>X'а под DOS.
- [4] Х. Партль, Э. Шлегль, И. Хина. *L<sup>A</sup>T<sub>E</sub>X: краткое описание*. — Пересказ с немецкого пособия [3] с дополнениями А. Шеня (shen@mcsme.ru). Рабочие материалы Независимого московского университета, 1993.
- [5] M. Spivak. *The Joy of T<sub>E</sub>X. A gourmet guide to typesetting with the A<sub>M</sub>S-T<sub>E</sub>X macro package*. — American Mathematical Society, Providence, RI, 1990. Русский перевод: М. Спивак. *Восхитительный T<sub>E</sub>X: руководство по комфортному изготовлению научных публикаций в пакете A<sub>M</sub>S-T<sub>E</sub>X*. — М.: Мир, 1993.
- [6] M. Goossens, F. Mittelbach, A. Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. — Addison-Wesley, 1994. Русский перевод: М. Гуссенс, Ф. Миттельбах, А. Самарин. *Путеводитель по пакету L<sup>A</sup>T<sub>E</sub>X и его расширению L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>*. 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 T<sub>E</sub>X. A simple introduction to A<sub>M</sub>S-L<sup>A</sup>T<sub>E</sub>X*. — Birkhäuser, 1993.
- [8] G. Grätzer. *First steps in L<sup>A</sup>T<sub>E</sub>X*. Birkhäuser; Springer-Verlag, 1999. (ISBN 0-8176-4132-7, 3-7643-4132-7) Перевод: Г. Грэтцер. *Первые шаги в L<sup>A</sup>T<sub>E</sub>X'e*. 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>*. — Новосибирск: Сибирский хронограф, 1998.
- [10] М. Гуссенс, С. Ратц. *Путеводитель по пакету L<sup>A</sup>T<sub>E</sub>X 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  $\text{\LaTeX}$  и его графическим расширениям. Иллюстрирование документов при помощи  $\text{\TeX}$ 'а и  $\text{\PostScript}$ 'а*. Перевод А. И. Лебедева и К. В. Мусатова под редакцией А. И. 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.

# Предметный указатель

`\"` 94  
`\'` 94  
— в окружении `tabbing` 190  
`\+` 190  
`\,` 80, 90  
`\-` 105, 106  
— в окружении `tabbing` 191  
`\.` 94  
`\/` 101  
`\:` 80  
`\;` 80  
`\<` 191  
`\=` 94  
— в окружении `tabbing` 186  
`\>` 186  
`\@` 93  
`\addtoreset` 272  
`\afterindentfalse` 282  
`\afterindenttrue` 282  
`\@chapapp` 281  
`\@dotsep` 288  
`\@dottedtocline` 287  
`\@evenfoot` 300  
`\@evenhead` 300  
`\@idxitem` 316  
`\@listI` 296  
`\@listi` 295  
`\@listii` 295  
`\@listiii` 295  
`\@listiv` 295  
`\@listv` 295  
`\@listvi` 295  
`\@makecaption` 309  
`\@makechapterhead` 280  
`\@makefnmark` 315  
`\@makefntext` 316  
`\@makeschapterhead` 281  
`\@oddfoot` 300  
`\@oddhead` 300  
`\@pnumwidth` 288  
`\@removefromreset` 272  
`\@startsection` 277  
`\@tocrmarg` 288  
`\[` 22  
`\` (backslash с пробелом) 13, 93  
`\!` 80  
`\#` 11  
`\$` 11  
`\%` 11  
`\\` 63, 66, 108, 192  
— в абзаце 108  
— в матрицах 63, 66  
— в окружении `tabbing` 186  
— в окружении `tabular` 192  
`\\*` 109  
`\_` 11  
`\{` 11, 24, 58  
`\}` 11, 24, 58  
`\|` 49  
`\]` 22  
`\^` 94  
`\‘` 94  
— в окружении `tabbing` 190  
`\~` 94  
`11pt` (классовая опция) 143  
`12pt` (классовая опция) 17, 143  
`\a` 188, 190  
`a4paper` (классовая опция) 144  
`a5paper` (классовая опция) 144  
`\AA` 94

- \aa 94
- \abovecaptionskip 309
- \abovedisplayshortskip 319
- \abovedisplayskip 319
- abstract (окружение) 154
  - в AMS'овских классах 354
- \abstractname 154
- \Acute 63
- \acute 62
- \addcontentsline 286
- \address 353
- \addtocontents 285
  - конфликт с \include 286
- \addtocounter 227
- \addtolength 149, 241
- \addvspace 218
- \AE 94
- \ae 94
- \afterepigraphskip 158
- \aleph 49
- align (окружение) 73, 74
- align\* (окружение) 74
- aligned (окружение) 75
- \Alph 228
- \alph 228
- \alpha 39
- \amalg 40
- amsart (класс документов) 143, 353
- amsbook (класс документов) 143, 353
- amscd (стилевой пакет) 77, 353
- amsfonts (стилевой пакет) 17, 353
- amsmath (стилевой пакет) 38, 46–48, 52, 55, 61, 67, 68, 70, 72, 75, 81, 142, 353
- amspc (класс документов) 143, 353
- amssymb (стилевой пакет) 38, 40–42, 44, 50, 51, 55, 61, 353
- $\mathcal{A}$ MS-TEX 322
- amsthm (стилевой пакет) 238, 353
- \and 158
- \angle 49, 50
- \appendix 154
- \appendixname 154
- \approx 41
- \approxeq 43
- \arabic 227
- \arccos 45
- \arcctg 45
- \arcsin 45
- \arctan 45
- \arctg 45
- \arg 45
- array (окружение) 66, 191, 198, 201, 203, 247, 255
- array (стилевой пакет) 205
- \arraycolsep 201
- \arrayrulewidth 201
- \arraystretch 203
- article (класс документов) 123, 143–146, 153, 158, 184, 233, 272–275, 291, 308, 318
- \Asbuk 228
- \asbuk 228
- \ast 51
- \asymp 41
- at-выражение 201
- \atop 69
- \author 158, 369
  - в AMS'овских классах 353
- aux-файл 21, 184
- \b 94
- b5paper (классовая опция) 144
- babel (стилевой пакет) 45, 90, 347, 349
- \backepsilon 43
- \backprime 50
- \backsim 43
- \backsimeq 43
- \backslash 49, 58
- badness 111
- \Bar 63
- \bar 62
- \barwedge 40
- \baselineskip 128, 149
- \baselinestretch 131
- \Bbbk 51
- beamer (класс документов) 369, 370
- \because 42
- \beforeepigraphskip 158
- \belowcaptionskip 309
- \belowdisplayskip 319

- `\belowdisplayskip` 319
- `\beta` 39
- `\beth` 51
- `\between` 42
- `\bfseries` 15, 100
- bib-файл 327, 329
- `\bibitem` 160, 183
  - с необязательным аргументом 161
- `\bibliography` 327
- `\bibliographystyle` 327
- `\bibname` 154
- BibTeX327
- `\bigcap` 48
- `\bigcirc` 40
- `\bigcup` 48
- `\Bigl` 60
- `\bigl` 60
- `\Bigr` 60
- `\bigr` 60
- `\Bigl` 60
- `\bigl` 60
- `\bigodot` 48
- `\bigoplus` 48
- `\bigotimes` 48
- `\Bigr` 60
- `\bigr` 60
- `\bigskip` 129
- `\bigskipamount` 109
- `\bigsqcup` 48
- `\bigstar` 50
- `\bigtriangledown` 40
- `\bigtriangleup` 40
- `\biguplus` 48
- `\bigvee` 48
- `\bigwedge` 48
- `\binom` 68
- `\binoppenalty` 53, 54
- `\blacklozenge` 50
- `\blacksquare` 50
- `\blacktriangle` 50
- `\blacktriangledown` 50
- `\blacktriangleleft` 42
- `\blacktriangleright` 42
- block (окружение) 370
- `\bmatrix` (окружение) 64
- `\boldsymbol` 55
- book (класс документов) 143–145, 158, 231, 273–275, 282, 287–289, 308, 317
- `\bot` 49
- `\botfigrule` 313
- `\bottomfraction` 311
- bottomnumber (счетчик) 310
- `\bowtie` 41
- `\boxdot` 40
- `\boxed` 61
- `\boxminus` 40
- `\boxplus` 40
- `\boxtimes` 40
- `\Breve` 63
- `\breve` 62
- bst-файл 334
- `\bullet` 40
- `\Bumpeq` 43
- `\bumpeq` 43
- `\c` 94
- `\Cap` 40
- `\cap` 40
- `\caption` 175, 176, 308
  - в окружении `longtable` 212
  - с необязательным аргументом 178
- caption (стилевой пакет) 310
- cases (окружение) 75
- CD (окружение) 77
- `\cdot` 40
- `\cdots` 25
- center (окружение) 18, 116, 292
- `\centerdot` 40
- `\cfrac` 70
- `\ch` 45
- `\chapter` 152, 277
- chapter (счетчик) 273–275
- `\Check` 63
- `\check` 62
- `\chi` 39
- `\choose` 69
- `\circ` 40
- `\circeq` 42
- `\circle` 373
- `\circle*` 373
- `\circlearrowleft` 44
- `\circlearrowright` 44



- \circledast 41
- \circledcirc 41
- \circleddash 41
- \circledS 50
- \cite 161, 183
  - с необязательным аргументом 161
- \cleardoublepage 127
- \clearpage 127
- \cline 196
- \clubpenalty 129
- \clubsuit 49
- \colon 42, 84
- \color 375
- color (стилевой пакет) 375
- \colorbox 375
- \columnsep 148
- \columnseprule 148
- \CompileMatrices 341
- \complement 50
- Computer Modern, шрифты 323
- \cong 41
- \contentsname 154, 291
- \coprod 48
- \copy 267
- \copyright 49, 90
- \cos 45
- \cosec 45
- \cosh 45
- \cot 45
- \coth 45
- \csc 45
- csf-файл 333
- CTAN 270, 367
- \ctg 45
- \cth 45
- \Cup 40
- \cup 40
- \curlyeqprec 42
- \curlyeqsucc 42
- \curlyvee 40
- \curlywedge 40
- \curraddr 353
- \curvearrowleft 44
- \curvearrowright 44
- \d 94
- \dag 49
- \dagger 40
- \daleth 51
- \dashrightarrow 51
- \dashv 41
- \date 158, 369
- \dbinom 68
- \dblfigrule 313
- \dblfloatpagefraction 312
- \dblfloatsep 312
- \dbltextfloatsep 312
- \dbltopfraction 312
- dbltopnumber (счетчик) 311
- \ddag 49
- \ddagger 40
- \Ddot 63
- \ddot 62
- \ddots 65
- \DeclareMathOperator 46
- \DeclareMathOperator\* 48
- \dedicatory 353
- \definecolor 376
- definition (стиль оформления теорем) 238
- \deg 45
- \Delta 39
- \delta 39
- depth (ключевое слово) 136, 263
- description (окружение) 117, 121, 292
- \det 48
- \dfrac 67
- \diagdown 50
- \diagup 50
- \diamond 40
- \diamondsuit 49
- \digamma 51
- \dim 45
- dirty tricks (грязные трюки) 80, 84
- displaymath (окружение) 145
- \displaystyle 82
- \div 40
- \divideontimes 41
- \documentclass 14, 142
- \Dot 63
- \dot 62
- \doteq 41
- \doteqdot 42, 51

- \dotfill 258
- \dotplus 40
- \dots 90
- \doublebarwedge 40
- \doublecap 51
- \doublecup 51
- \doublehyphendemerits 114
- \Downarrow 44
- \downarrow 44
- \downdownarrows 44
- \downharpoonleft 42
- \downharpoonright 42
- \dp 268
- draft (классовая опция) 144, 174
- dtx-файл 368, 377
- dvi-драйвер 10, 323
- dvi-файл 10
- \ell 49
- \email 353
- \emergencystretch 107, 112
- \emph 96
- empty (стиль оформления страниц) 146
- \emptyset 49
- \endfirsthead 211
- \endfoot 212
- \endgraf 159
- \endhead 211
- \endinput 28
- \endlastfoot 212
- \enlargethispage 128
- \enskip 92
- \ensuremath 216, 225
- enumerate (окружение) 117, 119–121, 234, 235, 292, 391
- enumi (счетчик) 235, 276
- enumii (счетчик) 276
- enumiii (счетчик) 276
- enumiv (счетчик) 276
- \epigraph 156
- epigraph (стилевой пакет) 156
- epigraphs (окружение) 157
- \epigraphsize 157
- \epigraphwidth 157
- eps (Encapsulated PostScript) 174
- \epsilon 39
- \eqcirc 42
- eqnarray (окружение) 76, 77, 145
- eqnarray\* (окружение) 77
- \eqno 52
- \eqref 52
- \eqsim 43
- \eqslantgtr 42
- \eqslantless 42
- equation (окружение) 52, 145
- equation (счетчик) 275
- \equiv 41
- \eta 39
- \eth 51
- \evensidemargin 148
- executivepaper (классовая опция) 144
- \exhyphenpenalty 114
- \exists 49
- \exp 45
- \extrarowheight 205
- \fallingdotseq 42
- fancyhdr (стилевой пакет) 300
- \fbox 91
- \fboxrule 252
- \fboxsep 252
- fdd-файл 377
- figure (окружение) 175–177 — с необязательным аргументом 177
- figure (счетчик) 275
- figure\* (окружение) 177
- \figurename 154, 175
- fil (единица измерения клея) 261
- \fill 131
- fill (единица измерения клея) 261
- \finalhyphendemerits 115
- \Finv 51
- \firstline 206
- \flat 49
- fleqn (классовая опция) 145
- \floatpagefraction 311
- \floatsep 312
- \flushbottom 134
- flushleft (окружение) 116, 292
- flushright (окружение) 116, 292
- \fnsymbol 228
- fontenc (стилевой пакет) 346
- \footnote 123

- 
- footnote (счетчик) 272, 275
  - \footnotemark 124
  - \footnoterule 314
  - \footnotesep 315
  - \footnotesize 97
  - \footnotetext 124
  - \footskip 301
  - \forall 49
  - \foreignlanguage 350
  - \frac 24
  - frame (окружение) 370
  - \framebox 252
  - \frown 41
  - \fussy 107
  
  - \Game 51
  - \Gamma 39
  - \gamma 39
  - gather (окружение) 73
  - gather\* (окружение) 73
  - gathered (окружение) 75
  - \gcd 48
  - \ge 23, 41
  - \genfrac 68
  - geometry (стилевой пакет) 151
  - \geq 51
  - \geqq 42
  - \geqslant 41, 42
  - \gets 44
  - \gg 41
  - \ggg 43
  - \gggtr 51
  - \gimel 51
  - \gnapprox 43
  - \gneq 43
  - \gneqq 43
  - \gnsim 43
  - graphicx (стилевой пакет) 172
  - \Grave 63
  - \grave 62
  - \gtrapprox 42
  - \gtrdot 41
  - \gtreqless 42
  - \gtreqqless 42
  - \gtrless 42
  - \gtrsim 42
  - \gvertneqq 43
  
  - \H 94
  - \hangafter 138
  - \hangindent 138
  - \Hat 63
  - \hat 62, 63
  - \hbar 49, 51
  - \hbox 255
  - \hdotsfor 65
  - \headheight 149, 301
  - headings (стиль оформления страниц) 146
  - \headsep 149, 301, 302
  - \heartsuit 49
  - \height 251
  - height (ключевое слово) 136
  - \hfil 257
  - \hfill 258
  - \hfuzz 111
  - \hhline 208
  - hhline (стилевой пакет) 208
  - \hline 193
  - \hoffset 151
  - \hom 45
  - \hookleftarrow 44
  - \hookrightarrow 44
  - \hphantom 83
  - \href 335
  - \hrule 135
  - \hrulefill 258
  - \hslash 51
  - \hspace 92, 131
  - \hspace\* 93
  - \hss 262
  - \ht 268
  - \Huge 97
  - \huge 97
  - hyperref (стилевой пакет) 335, 351, 352
  - \hyphenation 105
  - \hyphenpenalty 114
  
  - \i 94
  - idx-файл 163
  - \iiiint 48
  - \iiint 48
  - \iint 48
  - \Im 49
  - переопределение 46

- `\imath` 49
- `\in` 41
- `\include` 28, 184
  - конфликт с `\addtocontents` 286
- `\includegraphics` 173
- `\includeonly` 28
- `\index` 163
- `\indexentry` 164
- `\indexname` 154
- `\indexspace` 164
- `\inf` 48
- `\infty` 49
- `\injl` 48
- `\input` 27
- `inputenc` (стилевой пакет) 162, 349
- `\inst` 369
- `\institute` 369
- `\int` 48
- `\intercal` 41
- `\intertext` 74
- `\intertextsep` 312
- `\iota` 39
- `\item` 117
  - в окружении `theindex` 164
  - квадратная скобка после команды 119
  - необязательный аргумент 118, 121
- `itemize` (окружение) 117, 118, 122, 139, 234, 292
- `\itemsep` 294
- `\itshape` 100
- `\j` 94
- `\jmath` 49
- `\kappa` 39
- `\ker` 45
- `\keywords` 353
- `\kill` 187
- Knuth, Donald E. 6, 9, 320
- `\L` 94
- `\l` 94
- `l@-команда` 286
- `\label` 20, 73, 181, 183
- `\labelenumi` 235
- `\labelenumii` 235
- `\labelenumiii` 235
- `\labelenumiv` 235
- `\labelitemi` 234
- `\labelitemii` 234
- `\labelitemiii` 234
- `\labelitemiv` 234
- `\labelsep` 293
- `\labelwidth` 293
- `\Lambda` 39
- `\lambda` 39
- Lamport, Leslie 9
- `\land` 51
- `landscape` (классовая опция) 144
- `\langle` 58
- `\LARGE` 97
- `\Large` 97
- `\large` 97
- `\lasthline` 206
- `\LaTeX` 12
- `layout` (стилевой пакет) 149
- `\lbrace` 51, 165
- `\lbrack` 51
- `\lceil` 58
- `\ldotp` 84
- `\ldots` 25, 90
- `\le` 23, 41
- `\leaders` 258, 268
- `\left` 24, 58
- `\Leftarrow` 44
- `\leftarrow` 51
- `\leftarrowtail` 44
- `\lefteqn` 79, 80, 83
- `\leftharpoondown` 44
- `\leftharpoonup` 44
- `\leftleftarrows` 44
- `\leftmargin` 293
- `\leftmark` 303
- `\Leftrightarrow` 44
- `\leftrightarrow` 44
- `\leftrightharpoons` 44
- `\leftrightharpoons` 42
- `\leftrightsquigarrow` 44
- `\leftskip` 290
- `\leftthreetimes` 40
- `legalpaper` (классовая опция) 144
- `\leq` 51
- `\leqno` 53

- leqno (классовая опция) 145
- \leqq 42
- \leqslant 41, 42
- \lessapprox 42
- \lessdot 41
- \lesseqgtr 42
- \lesseqqgtr 42
- \lessgtr 42
- \lesssim 42
- letterpaper (классовая опция) 144
- \lfloor 58
- \lg 45
- \lim 48
- \liminf 48
- \limits 49
- \limsup 48
- \line 372
- \linebreak 108
- с необязательным аргументом 109
- list (окружение) 297–299
- \listfigurename 154
- \listoffigures 178
- \listoftables 178
- \listparindent 294
- \listtablename 154
- \ll 41
- \llcorner 59
- \Lleftarrow 44
- \lll 43
- \lless 51
- \ln 45
- \lnapprox 43
- \lneq 43
- \lneqq 43
- \lnot 51
- \lnsim 43
- lof-файл 178, 284
- \log 45
- log-файл 29, 103
- \Longleftarrow 44
- \longleftarrow 44
- \Longleftrightharrow 44
- \longleftrightharrow 44
- \longmapsto 44
- \Longrightarrow 44
- \longrightarrow 44
- longtable (окружение) 210, 211
- longtable (стилевой пакет) 17, 210
- \looparrowleft 44
- \looparrowright 44
- \looseness 113
- \lor 51
- lot-файл 178, 284
- \lozenge 50
- \lrcorner 59
- \Lsh 42
- \LTcapwidth 212
- \ltimes 41
- \lvertneqq 43
- \makeatletter 271
- \makeatother 271
- \makebox 248, 249
- \makeindex 163
- makeindex (программа) 162–171
- стилевой файл 169
- \MakeLowercase 305
- \maketitle 158
- \MakeUppercase 305
- \mapsto 44
- \marginpar 184
- с необязательным аргументом 185
- \marginparpush 185
- \marginparsep 185
- \marginparwidth 185
- \markboth 303
- \markright 303, 304
- \mathbb 55
- \mathbf 54, 57
- \mathbin 85
- \mathcal 57
- \mathfrak 56
- \mathop 85
- \mathrm 57
- \mathsf 57
- \mathstrut 83
- \mathsurround 87, 234, 316
- \mathtt 57
- matrix (окружение) 64
- \max 48
- MaxMatrixCols (счетчик) 64
- \mbox 108, 124, 247
- в формулах 58

- для предотвращения переноса 108
- как «пустой текст» на странице 128
- `\mdseries` 15, 100
- `\measuredangle` 50
- `\medskip` 129
- `\medskipamount` 109
- METAFONT 323
- `\mho` 51
- microtype (стилевой пакет) 112
- `\mid` 41
- `\min` 48
- minipage (окружение) 252
- minus (ключевое слово) 130, 131, 260
- `\mod` 47
- `\models` 41
- `\mp` 40
- `\mu` 39
- multicol (стилевой пакет) 133, 177, 318
- multicols (окружение) 133, 318
- `\multicolumn` 195
- `\multimap` 42
- multline (окружение) 72
- multline\* (окружение) 72
- `\multlinegap` 72
- myheadings (стиль оформления страниц) 146, 307
- `\nabla` 49
- `\natural` 49
- `\ncong` 43
- `\ne` 41
- `\nearrow` 44
- `\neg` 49
- `\neq` 51
- `\newcommand` 214, 222
- `\newcounter` 226
- с необязательным аргументом 230
- `\newenvironment` 243
- `\newenvironment*` 245
- `\newlength` 240
- `\newpage` 127, 128
- в окружении longtable 212
- `\newsavebox` 266
- `\newtheorem` 236
- `\newtheorem*` 238
- `\nexists` 50
- `\ngeq` 43
- `\ngeqq` 43
- `\ngeqslant` 43
- `\ngtr` 43
- `\ni` 41
- `\nLeftarrow` 44
- `\nleftarrow` 44
- `\nLeftrightarrow` 44
- `\nleqtriarightarrow` 44
- `\nleq` 43
- `\nleqq` 43
- `\nleqslant` 43
- `\nless` 43
- `\nmid` 43
- `\nocite` 332
- `\noindent` 126, 138
- `\nolimits` 49
- `\nolinebreak` 109
- `\nonumber` 76
- `\nopagebreak` 127
- в окружении longtable 212
- `\normalfont` 100
- `\normalmarginpar` 185
- `\normalsize` 97, 296
- `\not` 61
- `\notag` 73, 74
- `\notin` 41, 61
- notitlepage (классовая опция) 158
- `\nparallel` 43
- `\nprec` 43
- `\npreceq` 43
- `\nrightarrow` 44
- `\nrightharpoonright` 44
- `\nshortmid` 43
- `\nshortparallel` 43
- `\nsim` 43
- `\nsubseteq` 43
- `\nsubseteqq` 43
- `\nsucc` 43
- `\nsucceq` 43
- `\nsupseteq` 43
- `\nsupseteqq` 43
- `\ntriangleleft` 44
- `\ntrianglelefteq` 44

- `\ntriangleright` 44
- `\ntrianglerighteq` 43
- `\nu` 39
- `\numberwithin` 273
- `\nVDash` 43
- `\nVdash` 43
- `\nvDash` 43
- `\nvdash` 43
- `\narrow` 44
- `\O` 94
- `\o` 94
- `\oddsidemargin` 148
- `\odot` 40
- `\OE` 94
- `\oe` 94
- `\oint` 48
- `\Omega` 39
- `\omega` 39
- `\ominus` 40
- `onecolumn` (классовая опция) 144
- `oneside` (классовая опция) 144
- `\only` 370
- `openany` (классовая опция) 145, 282
- `openright` (классовая опция) 145, 282
- `\oplus` 40
- `\oslash` 40
- `OT1` (кодировка) 343
- `otherlanguage` (окружение) 350
- `\otimes` 40
- `\oval` 373
- с необязательным аргументом 373
- `\overbrace` 71
- `overfull` 103, 261
- в колонтитуле 152
- `\overleftarrow` 63
- `\overline` 62
- `\overrightarrow` 62
- `\owns` 51
- `\P` 49
- `\pagebreak` 128
- в окружении `longtable` 212
- `\pagenumbering` 146
- `\pageref` 20, 181, 182
- `\pagestyle` 146, 301
- `\par` 125
- `\paragraph` 152
- `paragraph` (счетчик) 231
- `\parallel` 41
- `\parbox` 250
- с необязательным аргументом 251
- `\parfillskip` 113, 114, 290
- `\parsep` 294
- `\parshape` 139
- `\parskip` 132
- `\part` 152, 277
- `\partial` 49
- `\partname` 154
- `\partopsep` 294
- `\pause` 370
- `\pdfoutput` 324
- `\perp` 41
- `\phantom` 83
- `\Phi` 39
- `\phi` 39
- `\Pi` 39
- `\pi` 39
- `picture` (окружение) 175, 255, 371, 372
- `\pitchfork` 43
- `plain` (стиль оформления страниц) 146
- `plain` (стиль оформления теорем) 238
- `plus` (ключевое слово) 130, 131, 260
- `\pm` 40
- `pmatrix` (окружение) 63
- `\pod` 47
- `\poptabs` 189
- `\pounds` 49, 90
- `\Pr` 48
- `\prec` 41
- `\precapprox` 43
- `\preccurlyeq` 42
- `\preceq` 41
- `\precnapprox` 43
- `\precneqq` 43
- `\precnsim` 43
- `\precsim` 42
- `\prime` 49

- `proc` (класс документов) 143–145,  
 153, 158, 274, 275  
`\prod` 48  
`\projlim` 48  
`proof` (окружение) 239  
`\proofname` 239  
`\propto` 41  
`\protect` 155, 285  
 — в аргументе `\addcontentsline`  
 287  
 — в аргументе `\addtocontents` 285  
 — в пометке 308  
`\Psi` 39  
`\psi` 39  
`\pushtabs` 189  
`\put` 372  
  
`\qbezier` 373  
`\qed` 239  
`\qitem` 157  
`\qqquad` 57, 80, 92  
`\quad` 80, 81, 92  
`quotation` (окружение) 116, 292  
`quote` (окружение) 115, 116, 139,  
 292  
  
`\r` 94  
`\raggedbottom` 134  
`\raggedright` 110  
`\raisebox` 253  
`\rangle` 58  
`\rbrace` 51, 165  
`\rbrack` 51  
`\rceil` 58  
`\Re` 49  
 — переопределение 46  
`\ref` 21, 181–183  
 — в окружении `enumerate` 120  
 — ссылка на плавающую  
 иллюстрацию 176  
 — ссылка на счетчик,  
 определенный пользователем  
 232  
 — ссылки на раздел документа 152  
 — ссылки на формулы 52  
`\refname` 154  
`\refstepcounter` 230  
`\relax` 136  
  
`\relpenalty` 53, 54  
`remark` (стиль оформления теорем)  
 238  
`remreset` (стилевой пакет) 272  
`\renewcommand` 154, 219  
`\renewcommand*` 222  
`\renewenvironment` 245  
`report` (класс документов)  
 143–145, 158, 184, 233, 274,  
 275, 282, 308, 317  
`\reversemarginpar` 185  
`\rfloor` 58  
`\rho` 39  
`\right` 24, 58  
`\Rightarrow` 44  
`\rightarrow` 51  
`\rightarrowtail` 44  
`\rightharpoondown` 44  
`\rightharpoonup` 44  
`\rightleftarrows` 44  
`\rightleftharpoons` 42, 44  
`\rightmargin` 293  
`\rightmark` 303  
`\rightrightarrows` 44  
`\rightskip` 290  
`\rightsquigarrow` 44  
`\rightthreetimes` 40  
`\risingdotseq` 42  
`\rlap` 263  
`\rmfamily` 100  
`\Roman` 227  
`\roman` 227  
`\Rrightarrow` 44  
`\Rsh` 42  
`\rtimes` 41  
`\rule` 134  
 — с необязательным аргументом  
 134  
  
`\S` 49, 90  
`\savebox` 267  
`\sbox` 267  
`\scriptscriptstyle` 82  
`\scriptsize` 97  
`\scriptstyle` 82  
`\scshape` 100  
`\searrow` 44  
`\sec` 45



- secdot (стилевой пакет) 283
- secnumdepth (счетчик) 276, 277, 281, 304
- \section 151, 277, 370
  - в AMS'овских классах 354
  - с необязательным аргументом 151
- section (счетчик) 229, 273, 274
- \section\* 152
- \sectionmark 304
- \selectfont 132, 346
- \selectlanguage 350
- \setcounter 227
- \setlength 149
- \setminus 40
- \settodepth 242
- \settoheight 242
- \settowidth 180
- \settowidth 242
- \sffamily 100
- \sh 45
- \sharp 49
- \shortmid 43
- \shortparallel 43
- \shoveleft 72
- \shoveright 73
- showkeys (стилевой пакет) 183
- \Sigma 39
- \sigma 39
- \sim 41
- \simeq 41
- \sin 45
- \sinh 45
- \slash 106, 114
- \sloppy 106, 112
- \slshape 13, 15, 100
- \small 97
- \smallfrown 43
- smallmatrix (окружение) 66, 67
- \smallsetminus 41
- \smallskip 129
- \smallskipamount 109
- \smallsmile 43
- \smash 83
- \smile 41
- \sourceflush 158
- \spadesuit 49
- \specialsection 354
- \sphericalangle 50
- split (окружение) 73
- \sqcap 40
- \sqcup 40
- \sqrt 25
- \sqsubset 42
- \sqsubseteq 41
- \sqsupset 42
- \sqsupseteq 41
- \square 50
- \ss 94
- \stackrel 70, 79
- \star 40
- \stepcounter 231
- \stop 34
- \strut 136, 137, 203, 302
- \subitem 164, 316
- \subjclass 353
- \subparagraph 152
- subparagraph (счетчик) 231
- \subsection 152, 277, 370
  - в AMS'овских классах 354
- \subsectionmark 305
- \Subset 43
- \subset 41
- \subseteq 41
- \subseteqeq 43
- \subsetneq 43
- \subsetneqq 43
- \substack 70
- \subsubitem 164, 316
- \subsubsection 152, 370
  - в AMS'овских классах 354
- subsubsection (счетчик) 231
- \succ 41
- \succapprox 43
- \succcurlyeq 42
- \succeq 41
- \succapprox 43
- \succneqq 43
- \succnsim 43
- \succsim 42
- \sum 48
- \sup 48
- \suppressfloats 181
- \Supset 43

- `\supset` 41
- `\supseteq` 41
- `\supseteqq` 43
- `\supsetneq` 43
- `\supsetneqq` 43
- `\surd` 49
- `\swarrow` 44
- `\symbol` 102, 123
- `\t` 94
- T1 (кодировка) 344
- T2A (кодировка) 345
- tabbing (окружение) 186, 292
- `\tabcolsep` 201
- table (окружение) 177, 276
- table (счетчик) 275
- table\* (окружение) 177, 178
- `\tablename` 154, 177
- `\tableofcontents` 160
  - стандартное определение 291
- tabular (окружение) 191–208, 247, 255
  - с необязательным аргументом 193
- `\tabularnewline` 197
- `\tan` 45
- `\tanh` 45
- `\tau` 39
- `\tbinom` 68
- tex-файл 348
- `\TeX` 12
- `\text` 57
- `\textbf` 96, 100
- `\textcolor` 375
- textcomp (стилевой пакет) 90, 235
- `\textfloatsep` 312
- `\textflush` 157
- `\textfraction` 311
- `\textheight` 149
- `\textit` 96, 100
- `\textmd` 100
- `\textnormal` 100
- `\textnumero` 90
- `\textrm` 100
- `\textsc` 97, 100
- `\textsf` 96, 100
- `\textsl` 96, 100
- `\textsterling` 90
- `\textstyle` 82
- `\texttt` 96, 100
- `\textup` 97, 100
- `\textwidth` 147
- TeX'овское приглашение 33
- tfm-файл 323
- `\tfrac` 67
- `\tg` 45
- `\th` 45
- `\thanks` 159
  - в AMS'овских классах 353
- the-команда 233
- thebibliography (окружение) 160
- `\theindex` 317
- theindex (окружение) 164, 171, 316–318, 387
- `\theoremstyle` 238
- `\therefore` 42
- `\Theta` 39
- `\theta` 39
- `\thickapprox` 43
- `\thicksim` 43
- `\thispagestyle` 146
- tikz (стилевой пакет) 175
- `\Tilde` 63
- `\tilde` 62
- `\times` 40
- `\tiny` 97
- `\title` 158, 369
  - в AMS'овских классах 353
- titlepage (классовая опция) 158
- titlepage (окружение) 159
- titlesec (стилевой пакет) 284
- `\to` 44
- to (ключевое слово) 256
- toc-файл 160, 284
- tocdepth (счетчик) 277, 287
- `\tolerance` 112
- `\top` 49
- `\topfigrule` 313
- `\topfraction` 311
- `\topmargin` 148
- topnumber (счетчик) 310
- `\topsep` 294
- `\topskip` 149
- `\totalheight` 251
- totalnumber (счетчик) 311

- `\triangle` 49
- `\triangledown` 50
- `\triangleleft` 40
- `\trianglelefteq` 42
- `\triangleq` 42
- `\triangleright` 40
- `\trianglerighteq` 42
- `trivlist` (окружение) 299
- `\ttfamily` 100, 122
- `\twocolumn` 318
- `twocolumn` (классовая опция) 144
- `\twoheadleftarrow` 44
- `\twoheadrightarrow` 44
- `twoside` (классовая опция) 144, 145, 184
- 
- `\u` 94
- `\uchyph` 115
- `\ulcorner` 59
- `\underbrace` 71
- `underfull` 104, 261
  - при нехватке клея 256
  - при печати страницы 134
- `\underline` 91
- `\unitlength` 371
- `\Uparrow` 44
- `\uparrow` 44
- `\Updownarrow` 44
- `\updownarrow` 44
- `\upharpoonleft` 42
- `\upharpoonright` 42
- `\uplus` 40
- `\upshape` 15, 100
- `\Upsilon` 39
- `\upsilon` 39
- `\upuparrows` 44
- `\urcorner` 59
- `\url` 335
- `\usebox` 267
- `\usecounter` 298
- `\usepackage` 17, 142
  - с необязательным аргументом 142
- 
- `\v` 94
- `\varDelta` 39
- `\varepsilon` 39
- `\varGamma` 39
- `\varinjlim` 48
- `\varkappa` 41, 51
- `\varLambda` 39
- `\varliminf` 48
- `\varlimsup` 48
- `\varnothing` 50
- `\varOmega` 39
- `\varPhi` 39
- `\varphi` 39
- `\varPi` 39
- `\varpi` 39
- `\varprojlim` 48
- `\varpropto` 43
- `\varPsi` 39
- `\varrho` 39
- `\varSigma` 39
- `\varsigma` 39
- `\varsubsetneq` 43
- `\varsubsetneqq` 43
- `\varsupsetneq` 43
- `\varsupsetneqq` 43
- `\varTheta` 39
- `\vartheta` 39
- `\vartriangle` 42
- `\vartriangleleft` 42
- `\vartriangleright` 42
- `\varUpsilon` 39
- `\varXi` 39
- `\vbox` 265
- `\Vdash` 42
- `\vDash` 42
- `\vdash` 41
- `\vdots` 65
- `\Vec` 63
- `\vec` 62
- `\vector` 372
- `\vee` 40
- `\veebar` 40
- `\verb` 122, 214
  - в макроопределении 214
  - в сносах 123
- `\verb*` 123
- `verbatim` (окружение) 122, 123, 214
  - в макроопределении 214
  - в сносах 123
- `verbatim` (стилевой пакет) 123
- `verbatim*` (окружение) 123

- `\verbatiminput` 123
- `verse` (окружение) 117, 292
- `\Vert` 51
- `Vmatrix` (окружение) 64
- `vmatrix` (окружение) 64
- `\voffset` 151
- `\vphantom` 83
- `\vrule` 135, 207
- `\vspace` 130
- `\vspace*` 130
- `\Vvdash` 42
- `\wd` 268
- `\wedge` 40
- `\widehat` 62
- `\widetilde` 62
- `\widowpenalty` 129
- `\width` 249
- `width` (ключевое слово) 136
- `\wp` 49
- `\wr` 40
- `wrapfig` (стилевой пакет) 178, 180
- `wrapfigure` (окружение) 178, 179
- `wratable` (окружение) 178
- $\LaTeX$  325
- $\TeX$  325
- `\Xi` 39
- `\xi` 39
- `\xleftarrow` 71
- `\xrightarrow` 71
- `\xspace` 225
- `xspace` (стилевой пакет) 225
- `xy` (стилевой пакет) 337
- `\xymatrix` 337
- $\Xy-pic$  337–342
- `\zeta` 39
- абзацы 11, 102
  - абзацный отступ 19
  - верстка без выравнивания 110
  - дополнительный интервал между абзацами 132
  - изменение количества строк 113
  - неправильной формы 139
  - нестандартной формы 137
  - неточное выравнивание по правому краю 111
  - подавление отступа 126
  - сообщения о трудностях при верстке 103, 104
- автор документа 158
- амперсенд 192
- аргумент 16, 55, 95
  - необязательный 17
  - перемещаемый 155
- базисная линия (baseline) 246
- бинарные операции 40, 84, 85
- бинарные отношения 41, 84, 85
- биномиальные коэффициенты 68, 69
- блок (box) 246
- буквы греческие 39
  - наклонные 39
- группы 15
- грязные трюки (dirty tricks) 80, 84
- дефис 88
- диакритические знаки 94
  - в окружении `tabbing` 188
- длина
  - единицы измерения 19
  - — `em` 20
  - — `ex` 20
  - — дюйм 19
  - — пика 19
  - — пункт 19
  - — пункт Дидо 19
  - — цигеро 19
  - параметры 19, 240
  - — присваивание значений 240
  - — с коэффициентом 241
  - — сложение 241
- доказательство 239
- дроби 24
- дроби, цепные 69, 82
- заглавие документа 158
- заметки на полях 184
- иллюстрации 175
  - при наборе в две колонки 177
  - стандартное название 175
- индексы 22, 23
- интеграл 48

- двойной, тройной и т. д. 48, 81
- контурный 48
- интерлиньяж 98, 131–132
- кавычки 89
  - елочки 89
  - лапки 89
- кернинг 88
- клей 130, 259
  - бесконечно сжимаемый 262
- Кнут, Дональд 6, 9, 320
- колонтитулы 300
  - высота 301
  - интервал между колонтитулом и текстом 301
  - передача информации из текста 303
- команды 12
  - примитивные 219
  - пробел после имени 13
  - со звездочкой 18
  - хрупкие (fragile) 155
- комментарии 11
- коммутативные диаграммы 77, 337–342
- корень 25
- коррекция наклона 101
- кривая Безье
  - квадратичная 373
  - кубическая 360
- лигатуры 88
- лидеры 258
  - в оглавлении 288
  - использование блоковых переменных 268
- линейки 134
  - невидимые 136, 200, 203
  - — в формулах 83
- Лэмпорт, Лесли 9
- макропакет 321
- макросы 213
  - новые окружения 243
  - с аргументами 220
  - с необязательным аргументом 223
- маргиналии 184
- матрицы 63
  - преамбула 66
- метапост 355–366
- многоточие 90
  - в тексте 90
  - в формулах 25
- надстрочные знаки
  - в тексте 94
  - в формулах 61
- неразрывный пробел ~ 91
- оглавление 160
  - запись текста без номера страницы 285
  - запись текста с номером страницы 286
  - модификация оформления 284
  - стандартный заголовок 160
  - степень детализации 277
- ограничители 58
- окружения 17
  - типа «теорема» 235–239
- операторы типа суммы 48, 84, 85
- опции классовые 17, 143
- опции стилевые 142
- пакет стилевой 17
- переносы
  - в словах с дефисом 105, 106
  - в словах, начинающихся с прописной буквы 115
  - затруднение и запрет 114
  - предотвращение в данном слове 108
  - указание разрешенных мест
    - — глобальное 105
    - — локальное 105
- перечеркнутые символы 43–44, 61
- перечни
  - модификация
    - — заголовков `enumerate` 235
    - — заголовков `itemize` 234
  - нумерованные (`enumerate`) 119
  - общего вида (`list`) 297–299
  - примитивные (`trivlist`) 299
  - простейшие (`itemize`) 118
  - с заголовками (`description`) 121

- пика 19
- плавающие иллюстрации
  - при наборе в две колонки 177
  - стандартное название 175
- плавающие таблицы 177
  - стандартное название 177
- подчеркивание 91
- поля 147
  - верхнее и нижнее 148
  - левое и правое 148
- пометки (marks) 303
  - автоматическое внесение в текст 304
- преамбула документа 14
- предметный указатель 162
  - модификация оформления 316
  - стандартное заглавие 164
- промежутки
  - в формулах 80, 85
  - вертикальные 129, 130
  - горизонтальные 92, 93
- пункт 19
- пункт Дидо 19
- разделы
  - варианты со звездочкой 152
  - модификация оформления 277, 280, 282
  - подавление отступа в первом абзаце 153, 278
  - подавление отступа в первом абзаце главы 282
  - стиль оформления заголовка 278
  - уровень вложенности 276
- рамки 61, 91, 252
- режимы  $\TeX$ 'а 125
- скобки 24
  - горизонтальные 71
  - переменного размера 24, 58, 60
- Сноски
  - оформление номеров 315
- сноски 123
  - к тексту внутри блока 124
  - к титульному листу 159
  - линейка, отделяющая сноски от текста 314
  - оформление текста сноски 316
  - пробел между страницей и сносками 314
- список иллюстраций 178
- список литературы 160
  - стандартное заглавие 162
- список таблиц 178
- сравнения по модулю 46
- стандартные заголовки 153
- степени 22, 23
- стихи 117
- страницы
  - запрет разрыва 127
  - изменение размера отдельной полосы 128
  - принудительный разрыв 127
  - — при двустороннем наборе 127
  - — с выдачей плавающих иллюстраций 127
  - сдвиг как целого 151
  - стиль нумерации 146
  - стиль оформления 146
  - — модификация 300
- стрелки 44
  - изогнутые (пакет  $\Xy-pic$ ) 338, 340
  - надпись над стрелкой 71, 78
  - надпись под стрелкой 78
  - надпись сбоку от стрелки 78
  - надпись, разрывающая стрелку (пакет  $\Xy-pic$ ) 339
  - сложного начертания (пакет  $\Xy-pic$ ) 338, 341
- строки
  - жидкие 103
  - — равномерное увеличение разреженности 107
  - — снятие запретов 106
  - запрет разрыва 109
  - — неразрывный пробел 91
  - насильственный разрыв 108
  - — с выравниванием 108
  - разреженные 103
- счетчики 226
  - **the**-команда 233
  - выдача на печать 227, 228
  - определенные при начале трансляции 233

- переподчинение существующего счетчика 272
- подчинение 230
- присваивание значений 227
- сложение 227
- создание 226
- ссылочный префикс 273
- увеличение на шаг 230

#### таблицы

- абзац в графе 196
- — в пакете `array` 207
- вертикальные линейки 194
- горизонтальные линейки 193
- графы в несколько колонок 195
- интервал между колонками 201
- невидимые линейки в строках 203
- преамбула 191
- — `!`-выражение (в пакете `array`) 207
- — `<`- и `>`-выражения (в пакете `array`) 208
- — `at`-выражение 201
- — символ `|` 194
- разбиение преамбулы на колонки 197
- толщина линеек 201
- частичные горизонтальные линейки 196

#### Табуляция

- использование позиций 186
- предварительная установка позиций 187
- установка позиций 186

#### табуляция

- выравнивание по правому краю 190
- запоминание и вспоминание позиций 189
- сдвиг первой позиции 190
- типе 88

- длинное (`em-dash`) 88
- короткое (`en-dash`) 88
- титульный лист 158
- дополнительная информация 158
- оформление вручную 159
- сноски 159
- точка отсчета 246, 372

#### ударение 95

#### форматный файл 321

#### формулы

- включение текста 57–58
- внутритекстовые 22
- выключные 22, 145
- — знак препинания после формулы 25
- надстрочные знаки 61
- — в пакете `amsmath` 63
- нумерация 52, 53, 145, 273
- переносы 53

#### цепные дроби 69, 82

#### цитаты 115, 116

#### цицеро 19

#### шрифты

- Computer Modern 323
- кодировка (`encoding`) 343
- — OT1 343
- — T1 344
- — T2A 345
- математический курсив 22
- наклонный 13
- насыщенность (`series`) 99
- начертание (`shape`) 99
- полужирный 15
- семейство (`family`) 99
- штрихи (в формулах) 25, 50, 86

#### эпиграф 156

*Сергей Михайлович Львовский*

## НАБОР И ВЁРСТКА В СИСТЕМЕ L<sup>A</sup>T<sub>E</sub>X

Подписано в печать 10.07.2014 г. Формат 70×100/16. Бумага офсетная. Печать офсетная. Печ. л. 25. Тираж 2000 экз. Заказ №

Издательство Московского центра непрерывного математического образования  
119002, Москва, Большой Власьевский пер., 11. Тел. (499) 241-72-85.

Отпечатано в типографии ООО «ТДДС-СТОЛИЦА-8»  
тел. 8(495)363-48-86. <http://capitalpress.ru>