

Л^AT_EX: подробное описание

С.М. Львовский

Предварительная рабочая версия

Оглавление

I. Элементарное введение	7
1. Общие замечания	7
1.1. Что такое $\text{T}_\text{E}\text{X}$ и $\text{L}_\text{A}\text{T}_\text{E}\text{X}$	7
1.2. Достоинства и недостатки	8
1.3. Литература по $\text{T}_\text{E}\text{X}$	8
1.4. Как проходит работа с системой $\text{L}_\text{A}\text{T}_\text{E}\text{X}$	9
2. Основные понятия	9
2.1. Исходный файл	9
2.2. Спецсимволы	10
2.3. Команды и их задание в тексте	11
2.4. Структура исходного текста	12
2.5. Группы	13
2.6. Параметры	14
2.7. Команды с аргументами	15
2.8. Окружения	15
2.9. Звездочка после имени команды	16
2.10. Единицы длины	16
2.11. Автоматическая генерация ссылок	17
3. Набор формул в простейших случаях	18
3.1. Основные принципы	18
3.2. Степени и индексы	18
3.3. Дроби	19
3.4. Скобки	20
3.5. Корни	21
3.6. Штрихи и многоточия	21
3.7. Функции типа синус	22
4. Разбиение исходного файла на части	22
5. Обработка ошибок	23
6. Как читать дальше?	28
II. Как набирать формулы	30
1. Таблицы спецзнаков с комментариями	30
1.1. Операции, отношения и просто значки	30
1.2. Операции с пределами и без	32
1.3. Разное	34
2. Важные мелочи	35
2.1. Нумерация формул	35
2.2. Переносы в формулах	37

2.3.	Некурсивные шрифты в математической формуле	37
2.4.	Включение текста в формулы	38
2.5.	Скобки переменного размера	39
2.6.	Перечеркнутые символы	41
2.7.	Надстрочные знаки	42
2.8.	Альтернативные обозначения для математических формул	43
3.	Одно над другим	44
3.1.	Простейшие случаи	44
3.2.	Набор матриц	45
3.3.	Переносы в выключных формулах	49
4.	Тонкая настройка	50
4.1.	Пробелы вручную	51
4.2.	Размер символов в формулах	51
4.3.	Фантомы и прочее	53
4.4.	Снова об интервалах в формулах	54
4.5.	Дополнительные пробелы вокруг формул	57
III.	Набор текста	58
1.	Специальные знаки	58
1.1.	Дефисы, минусы и тире	58
1.2.	Кавычки	59
1.3.	Многоточие	59
1.4.	Прочие значки	59
1.5.	Подчеркивания, рамки	60
2.	Промежутки между словами	60
2.1.	Неразрывный пробел	60
2.2.	Промежуток промежутку рознь	61
2.3.	Установка промежутков вручную	62
3.	Диакритические знаки и прочее	63
4.	Переключение шрифтов	64
5.	Сноски	67
6.	Абзацы	68
6.1.	Переносы	70
6.2.	Команда <code>\sloppy</code> и параметр <code>\emergencystretch</code>	71
6.3.	Ручное управление разрывами строк	73
6.4.	Верстка абзацев без выравнивания и переносов	74
6.5.	Более тонкая настройка	75
7.	Между абзацами	78
7.1.	Понятие о режимах \TeX а	78
7.2.	Подавление абзацного отступа	79
7.3.	Вертикальные промежутки	79
7.4.	Интерлиньяж	81
7.5.	Управление разрывами страниц	81
7.6.	Набор в две колонки	83
7.7.	Заключительные замечания о разрывах страниц и вертикальных интервалах.	83
8.	Специальные абзацы	84

8.1.	Цитаты	84
8.2.	Центрирование, прижатие текста к краю	85
8.3.	Стихи	86
8.4.	Перечни	86
8.5.	Буквальное воспроизведение (<i>verbatim, verb</i>)	90
8.6.	Абзацы нестандартной формы	91
9.	Линейки	93
9.1.	Линейки в простейшем виде	93
9.2.	TeXовские команды для генерации линейек	94
9.3.	Невидимые линейки	95
IV.	Оформление текста в целом	97
1.	Стили	97
1.1.	Стиль оформления страницы	99
2.	Поля, размер страницы и прочее	100
2.1.	Ширина	100
2.2.	Высота	101
2.3.	Сдвиг страницы как целого	101
3.	Разделы документа	102
3.1.	Команда <code>\section</code>	102
3.2.	Какие бывают разделы документа	103
3.3.	Изменение стандартных заголовков	104
3.4.	Аннотация, приложение	104
4.	Титул, оглавление и пр.	105
4.1.	Титул	105
4.2.	Оглавление	106
4.3.	Список литературы	106
4.4.	Предметный указатель	108
4.5.	Перемещаемые аргументы и хрупкие команды	111
5.	Плавающие иллюстрации и таблицы	112
6.	Заметки на полях	113
V.	Псевдорисунки	115
1.	Создание псевдорисунка	115
2.	Отрезки и стрелки	117
3.	Окружности, круги и овалы	117
4.	Дополнительные возможности	118
5.	Параметры, регулирующие вид псевдорисунка	120
VI.	Верстка текста с выравниванием	121
1.	Имитация табулятора	121
1.1.	Элементарные средства	121
1.2.	Более сложные средства	123
2.	Верстка таблиц	125
2.1.	Простейшие случаи	126
2.2.	Более сложные случаи	128
3.	Примеры	130
4.	Заключительные замечания	136

VII. Создание новых команд	137
1. Макроопределения	137
1.1. Команды без аргументов	137
1.2. Команды с аргументами	142
2. Счетчики	143
2.1. Создание счетчиков и простейшие операции с ними	144
2.2. Отношение подчинения между счетчиками	146
2.3. Организация автоматических ссылок	148
2.4. Счетчики, которые определять не надо	152
2.5. Модификация оформления перечней	152
3. Параметры со значением длины	153
4. Создание новых окружений	156
4.1. Новые окружения: общий случай	156
4.2. Окружения типа «теорема»	157
VIII. Блоки	160
1. Текст состоит из блоков	160
2. ЛАТЭХовские команды для генерации блоков	161
2.1. Блоки из строк	161
2.2. Блоки из абзацев	163
2.3. Текст в рамке; комбинации блоков	164
2.4. Сдвиги относительно базисной линии	165
3. Команда <code>\hbox</code>	166
3.1. Растяжимые интервалы	167
3.2. Лидеры	169
3.3. Клей	170
3.4. Бесконечно сжимаемые интервалы	171
3.5. Еще раз о линейках	173
4. Команда <code>\vbox</code>	174
5. Блочные переменные	175
IX. Модификация стандартных стилей	178
1. Еще раз о стилях и стилевых опциях	178
2. Снова о счетчиках	179
3. Разделы документа	181
3.1. Что нумеровать и что включать в оглавление	181
3.2. Модификация команд, задающих разделы	182
4. Оглавление, список иллюстраций и прочее	186
5. Перечни общего вида	192
5.1. Параметры, влияющие на оформление перечней	192
5.2. Окружения <code>list</code> и <code>trivlist</code>	194
6. Колонтитулы	196
7. Разное	204
7.1. Теоремы, выключные формулы	204
7.2. Сноски	205
7.3. Библиография	206
7.4. Предметный указатель	207

Приложение А.О Т_ЕХовских шрифтах	209
Приложение Б. Опция <code>ruscorr.sty</code>	213
Приложение В.Новая схема выбора шрифтов (NFSS)	217
Приложение Г. Стандартные стили	219
1. Основные размеры	219
2. Счетчики	221
3. Разделы документа	223
4. Колонтитулы	224
5. Перечни	226

Предисловие

Настоящее пособие посвящено издательской системе \LaTeX , предназначенной для набора и верстки текстов с формулами. Оно возникло из попытки дополнить книгу [3], переведенную А. Шенем ([4]).

Выражаю глубокую благодарность А. Шеню, без многочисленных бесед и споров с которым эта книга никогда не была бы написана; он же любезно согласился отредактировать книгу. Выражаю также глубокую благодарность В.Д. Арнольду за большую дружескую поддержку.

Глава I.

Элементарное введение

1. Общие замечания

1.1. Что такое $\text{T}_\text{E}\text{X}$ и $\text{L}_\text{A}\text{T}_\text{E}\text{X}$

$\text{T}_\text{E}\text{X}$ (произносится «тех», пишется также «TeX») — это созданная замечательным американским математиком и программистом Дональдом Кнудом (Donald E. Knuth) система для верстки текстов с формулами. Сам по себе $\text{T}_\text{E}\text{X}$ 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 $\text{T}_\text{E}\text{X}$ представляет собой пакет макроопределений (макропакет) этого языка. $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ (произносится «латех» или «лэйтех», пишется также «LaTeX») — это созданная Лесли Лампортом (Leslie Lamport) издательская система на базе $\text{T}_\text{E}\text{X}$.

Прежде, чем углубиться в изучение собственно $\text{L}_\text{A}\text{T}_\text{E}\text{X}$, скажем несколько слов о других издательских системах на базе $\text{T}_\text{E}\text{X}$. Наряду с $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ ом, распространены также макропакеты Plain $\text{T}_\text{E}\text{X}$ и $\text{A}\text{M}\text{S}-\text{T}_\text{E}\text{X}$. Макропакет Plain $\text{T}_\text{E}\text{X}$ был разработан самим Дональдом Кнудом, рассматривавшим его в качестве платформы для построения более сложных систем; на практике он используется и как средство для обмена текстами (текст, подготовленный для Plain $\text{T}_\text{E}\text{X}$, сравнительно несложно переделать в исходный текст для того же $\text{L}_\text{A}\text{T}_\text{E}\text{X}$). Что касается $\text{A}\text{M}\text{S}-\text{T}_\text{E}\text{X}$, то эта издательская система сориентирована на важный, но узкий круг приложений: верстку статей для математических журналов, издаваемых Американским Математическим Обществом. Соответственно, в $\text{A}\text{M}\text{S}-\text{T}_\text{E}\text{X}$ е предусмотрено большое количество весьма изощренных возможностей для создания сложных математических формул, но при этом нет многих вещей, которые естественно было бы ожидать в издательских системах общего назначения (например, автоматической нумерации частей документа). $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ в этом отношении более гармоничен. Наконец, недавно появился $\text{A}\text{M}\text{S}-\text{L}_\text{A}\text{T}_\text{E}\text{X}$, призванный сочетать мощь $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ а как издательской системы и изощренные возможности набора формул, предоставляемые $\text{A}\text{M}\text{S}-\text{T}_\text{E}\text{X}$ ом. Эта система находится в процессе становления, и писать о ней книгу пока преждевременно. В приложении В сказано несколько слов об одной важной особенности $\text{A}\text{M}\text{S}-\text{L}_\text{A}\text{T}_\text{E}\text{X}$ а — так называемой «новой схеме выбора шрифтов».

Настоящая книга посвящена описанию $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ а версии 2.09. Эта версия, вышедшая в 1989 году, завершает серию усовершенствований и исправлений в $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ е, начавшуюся с момента его создания в 1984 году, и является в данный момент общепринятой. Недавно было объявлено, что в марте 1994 года должна выйти новая версия $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ а — версия 3.

1.2. Достоинства и недостатки

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

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

\TeX (и все издательские системы на его базе) неприхотлив к используемой технике: им вполне можно пользоваться, например, на компьютерах на базе 80286-процессора,¹ а исходные тексты можно готовить и на совсем уж примитивных машинах. С другой стороны, \TeX овские файлы обладают высокой степенью переносимости: Вы можете подготовить \LaTeX овский исходный текст на своем IBM PC, переслать его в издательство, и быть уверенными, что там Ваш текст будет правильно обработан и на печати получится в точности то же, что получилось у Вас при пробной печати на Вашем любимом матричном принтере (с той единственной разницей, что фотонаборный автомат даст текст более высокого качества). Благодаря этому обстоятельству \TeX стал очень популярен как язык международного обмена статьями по математике и физике.

Есть у \TeX а и недостатки. Первый из них (разделяемый \TeX ом со всеми другими издательскими системами) таков: он работает относительно медленно, занимает много памяти, а полученный результат нельзя напечатать на дешевом принтере. Вторая особенность \TeX а, которая может не понравиться тем, кто привык к редакторам наподобие Chiwriter — это то, что он не является системой типа WYSIWYG: работа с исходным текстом и просмотр того, как текст будет выглядеть на печати, — разные операции. Впрочем, благодаря этой особенности время на подготовку текста существенно сокращается.

Далее, хотя параметры оформления менять легко, создать принципиально новое оформление (новый «стиль») — непростое дело. Впрочем, \LaTeX предоставляет довольно широкие возможности для модификации стандартных стилей.

Наконец, переносимость \TeX овских текстов снижается, если в них предусмотрен импорт графических файлов (эта возможность в \TeX е зависит от его реализации).

1.3. Литература по \TeX у

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

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

Наконец, каноническое описание \LaTeX а — книга Лесли Лэмпорта [1]. К сожалению, в ней недостаточно освещен такой важный на практике вопрос, как модификация стандартных \LaTeX овских стилей. Видимо, вместе с новой версией \LaTeX а появится и новое издание.

¹Есть даже реализации \TeX а на IBM XT, но при работе на таких компьютерах производительность труда будет слишком низка.

Настоящее пособие в чем-то уже, чем книга [1], а в чем-то — шире: мы не упоминаем о некоторых экзотических средствах Л^AT_EXа, которые либо не слишком полезны на практике, либо дублируют аналогичные средства Plain T_EXа, но при этом рассказываем о многих полезных вещах, о которых в [1] не упоминается.

Первой книгой по Л^AT_EXу на русском языке была книга [4], представляющая собой выполненный А. Шенем перевод краткого руководства [3] вместе с дополнением переводчика, посвященным описанию одной из популярных реализаций T_EXа на IBM PC — системы emT_EX.

1.4. Как проходит работа с системой Л^AT_EX

В дальнейшем мы будем отмечать, какие свойства системы специфичны для Л^AT_EXа, а какие относятся вообще к T_EXу и ко всем издательским системам на его базе, но при первом чтении Вы можете об этих тонкостях не задумываться и воспринимать слова T_EX и Л^AT_EX как синонимы.

Все, что сказано в этом разделе, применимо не только к Л^AT_EXу, но и к любому другому макропакету для T_EXа, хотя мы для краткости будем говорить только про Л^AT_EX.

Для начала автор должен подготовить файл с текстом, оснащенным командами для Л^AT_EXа, который по традиции имеет расширение `.tex` (описанию того, как именно его готовить, и посвящена вся эта книга). Дальнейшая работа протекает в два этапа. Сначала надо обработать файл с помощью программы-транслятора; в результате получается файл с расширением `.dvi` (`device-independent` — не зависящий от устройства).

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

На самом деле повторений цикла будет больше, так как при синтаксических ошибках в исходном тексте транслятор будет выдавать сообщения об ошибках, которые приходится исправлять.

Когда Вы получаете систему Л^AT_EX, Вам необходимо четко уяснить для себя три вопроса:

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

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

2. Основные понятия

2.1. Исходный файл

Исходный файл для системы Л^AT_EX представляет собой собственно текст документа вместе со *специмолами* 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 (ASCII-файл). Это означает, что текст *не должен* содержать шрифтовых выделений, разбивки на страницы и т. п.

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

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

Слова разделяются пробелами,
а абзацы — пустыми строками.

Абзацный отступ в исходном
тексте оставлять не надо: он по-
лучается автоматически.

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

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

Как мог заметить читатель из этого примера, тратить время на форматирование исходного текста тоже незачем.

2.2. Спецсимволы

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

{ } \$ & # % _ ^ ~ \

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

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

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

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

Это пример.
Жил-был у бабушки серенький
козлик.

Это % глупый
% Лучше: поучительный <-----
пример.

Жил-был у бабушки
сере%
нький козлик.

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

Скажем вкратце о смысле остальных спецсимволов. Фигурные скобки ограничивают *группы* в исходном файле (см. стр. 13). Знак доллара ограничивает математические формулы (см. главу II). При наборе математических же формул используются знаки `_` и `^` («знак подчеркивания» и «крышка»). Знак `~` обозначает «неразрывный пробел» между словами (см. стр. 60). Со знака `\` начинаются все \TeX овские команды (см. раздел 2.3). Знаки `#` и `&` используются в более сложных конструкциях \TeX а, о которых сейчас говорить преждевременно.

Наконец, символы

`<` `>` `|`

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

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 стр. 10.

Команды второго типа состоят из `\` и последовательности букв, называемой *именем команды* (имя может состоять и из одной-единственной 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.

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

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

В чем разница между `$1` и `$ 1`?

В чем разница между
`\$1` и `\$ 1`?

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

38 попугаев.

`\sl38 попугаев.`

Подарок мартышке.

`\sl Подарок мартышке.`

Если бы мы написали `\slПодарок мартышке`, то при трансляции `TeX` зафиксировал бы ошибку (типичную для начинающих) и выдал сообщение о том, что команда `\slПодарок` 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 фигурной 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 строчке этого примера мы сознательно 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

`\documentstyle`

задающей стиль оформления документа. Пример:

`\documentstyle{book}`

Слово `book` в фигурных скобках указывает, что документ будет оформлен, как книга: все главы будут начинаться с нечетной страницы, текст будет снабжен колонтитулами некоторого определенного вида и т. п. Кроме стиля `book`, в стандартный комплект `LaTeX`а входят стили `article` (для оформления статей), `report` (нечто среднее между `article` и `book`) и `letter` (для оформления деловых писем так, как это принято в США). Чтобы задать оформление документа с помощью одного из этих стилей, надо в фигурных скобках после команды `\documentstyle` указать вместо `book` название требуемого стиля. Стандартные стили можно (а иногда и нужно) менять, можно создавать и новые стили, но пока что будем исходить из стандартных стилей.

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

```
\begin{document}
```

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

Заканчиваться файл должен командой

```
\end{document}
```

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

Следующий пример показывает минимальный \TeX -файл, составленный по всем правилам. Ничего интересного в результате его обработки не напечатается, но уж зато и сообщения об ошибках Вы не получите.

```
\documentstyle{article}
```

```
\begin{document}
```

Проба пера.

```
\end{document}
```

2.5. Группы

Вторым важнейшим понятием \TeX а является понятие *группы*. Чтобы понять, что это такое, рассмотрим пример.

При обработке \TeX ом исходного файла набор текста в каждый момент идет каким-то вполне определенным шрифтом (он называется текущим шрифтом). Изначально текущим шрифтом является «обычный» прямой шрифт (по-ученому он называется «roman»). Команда `\sl`, с которой мы уже столкнулись в разделе 2.3, переключает текущий шрифт на наклонный, а команда `\bf` — на полужирный:

Полужирный шрифт начнется со **следующего слова**; дальше **так и пойдет**. Теперь *наклонный шрифт*, и снова **полужирный**, до нового переключения.

Полужирный шрифт начнется со `\bf` следующего слова; дальше так и пойдет. Теперь `\sl` наклонный шрифт, `\bf` и снова полужирный, до нового переключения.

Но как же быть, если нужно печатать полужирным шрифтом не весь текст, а только его часть? Можно было бы включить в текст команду `\rm`, переключающую шрифт снова на «обычный». Но есть более простой способ: часть текста, которую Вы хотите оформить полужирным шрифтом, можно заключить в фигурные скобки, и дать команду `\bf` *внутри* этих скобок! Тогда сразу же после закрывающей фигурной скобки \TeX «забудет» про то, что шрифт переключался на полужирный, 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 `{\bf это}` слово; после скобок все идет, как прежде.

Сами по себе фигурные скобки не генерируют никакого текста и не влияют на шрифт; единственное, что они делают — это ограничивают *группу* внутри файла. Как правило, задаваемые командами Т_ЕXа 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 `\it` (она переключает шрифт на курсивный):

Сначала переключим шрифт на полужирный, затем на курсив; временно перейдем снова на полужирный; посмотрите, как восстановится шрифт после конца группы.

Сначала {переключим шрифт `\bf` на полужирный, затем на `\it` курсив; временно перейдем снова на `{\bf полужирный;}` посмотрите, как восстановится} шрифт после кон{ца г}руппы.

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

Трюк с постановкой пары скобок `{}` после имени команды, о котором шла речь на стр. 12 — тоже пример использования групп. В этом случае скобки ограничивают «пустую» группу; ставятся они в качестве не-букв, ограничивающих имя команды и при этом никак не влияющих на печатный текст.

Фигурные скобки в исходном тексте должны быть сбалансированы²: каждой открывающей скобке должна соответствовать закрывающая. Если Вы почему-либо не соблюли это условие, при трансляции Вы получите сообщение об ошибке.

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

2.6. Параметры

Наряду с текущим шрифтом, о котором уже шла речь, Т_ЕX в каждый момент обработки исходного текста учитывает значения различных 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_ЕX обозначаются аналогично командам: с помощью символа `\` («backslash»), за которым следует либо последовательность букв, либо одна не-буква. Например, `\parindent` обозначает в Т_ЕXе величину абзацного отступа; если нам понадобилось, чтобы абзацный отступ равнялся двум сантиметрам, можно написать так:

²Это не относится к скобкам, входящим в состав команд `\{` и `\}`.

```
\parindent=2cm
```

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 разобранным примере), или еще что-нибудь.

2.7. Команды с аргументами

Команды наподобие `\LaTeX` или, скажем, `\bf` действуют «сами по себе»; многим командам, однако, необходимо задать *аргументы*. Первый пример тому дает команда `\documentstyle`: слово, указываемое в фигурных скобках — ее аргумент; если его не указать, то произойдет ошибка. В `ЛATEX` аргументы команд бывают обязательные и необязательные. Обязательные аргументы задаются *в фигурных скобках*; если для команды предусмотрено наличие обязательных аргументов, она без них правильно работать не будет. У многих команд предусмотрены также и необязательные аргументы, которые влияют на работу команды, коль скоро они указаны, но при этом нормальная работа команды не нарушится и при их отсутствии. Необязательные аргументы задаются *в квадратных скобках*.

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

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

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 аргументы, могут быть пробелы, но не должно быть пустых строк.

2.8. Окружения

Еще одна важная конструкция `ЛATEX`а — это окружение (environment).

Окружение — это фрагмент файла, который начинается с текста

```
\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 ей окружения). Например:

<p>Все строки этого абзаца будут центрированы; переносов слов не будет, если только слово, как дезоксирибонуклеиновая кислота, не длинней строки.</p>	<pre>\begin{center} Все строки этого абзаца будут центрированы; переносов слов не будет, если только слово, как дезоксирибонуклеиновая кислота, не длинней строки. \end{center}</pre>
---	---

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

Важнейшим 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 `\it`, и при этом после команды `\end{center}` восстановился бы тот шрифт, что был перед окружением.

2.9. Звездочка после имени команды

В \TeX 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». После имени команды со звездочкой пробелы *не игнорируются*; если команда со звездочкой имеет аргументы, пробела между звездочкой и аргументами быть не должно.

2.10. Единицы длины

Многие параметры, используемые \TeX ом, являются размерами (пример тому мы видели в разделе 2.6); в нижеследующей таблице собраны единицы длины (кроме нескольких экзотических), которые можно использовать в \TeX е при задании размеров.

<code>pt</code>	пункт \approx 0.35 миллиметра
<code>pc</code>	пика = 12pt
<code>mm</code>	миллиметр
<code>cm</code>	сантиметр = 10 mm
<code>in</code>	дюйм = 25,4 mm

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

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

```
\parindent=0
```

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

Кроме перечисленных, в Т_EXе 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. В частности, расстояние в 1em на глаз обычно воспринимается как «один пробел».

2.11. Автоматическая генерация ссылок

Т_EX 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 указать страницу, на которой это место находится, написав «...как мы уже отмечали на стр. 99» или что-то в этом роде. Проблема, однако, в том, что заранее нельзя угадать, на какую страницу печатного текста попадет это место. Вместо того, чтобы гадать, можно сделать следующее:

- Пометить то место, на которое Вы хотите сослаться в дальнейшем (или предшествующем) тексте;
- В том месте текста, где Вы хотите поместить ссылку, поставить команду-ссылку на Вашу метку.

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

```
\label{wash}
```

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

Обязательно мойте руки перед едой, чтобы не заболеть.

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

Обязательно мойте

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

Как известно (см. \стр~\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 `~`, чтоб при печати сокращение «стр.» попало на ту же строку, что и номер страницы, и команду `\` (backslash с пробелом), чтобы на

печати пробел после сокращения «см.» не получился больше, чем надо. Подробности см. в разделах III.2.1 и III.2.2.

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 не печати или при просмотре увидите на месте своих ссылок вопросительные знаки. Дело в том, что в этот момент \LaTeX еще не знает значения Ваших меток: он только записывает информацию о них в специальный файл (с тем же именем, что у обрабатываемого файла, и расширением `.aux`); при следующем запуске он прочтет эту информацию и подставит ссылки. В промежутке между двумя запусками в файл могли быть внесены изменения, что может привести к сдвигу нумерации страниц. Если такие изменения действительно произошли, \LaTeX сообщит Вам об этом и попросит запустить программу еще раз, чтобы получить корректные ссылки.

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 главе IV.

3. Набор формул в простейших случаях

3.1. Основные принципы

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

Часть файла, составляющая математическую формулу, образует группу (см. стр. 13): изменения параметров, произведенные внутри формулы, забываются по ее окончании.

3.2. Степени и индексы

Степени и индексы набираются с помощью знаков \hat и $_$ соответственно.

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

Катеты a , b треугольника связаны с 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

$$x^{1993} + y^{1993} = z^{1993}$$

не имеет решений в натуральных числах.

Из теоремы Ферма следует, что уравнение

$$x^{1993} + y^{1993} = z^{1993}$$

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 один под другим, а на разных расстояниях от выражения, к которому они относятся, то нужно Т_ЕХ немного обмануть, оформив часть индексов как индексы к «пустой формуле» (паре из открывающей и закрывающей скобок):

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

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

Если Вы хотите написать формулу, читающуюся как «два в степени x в кубе», то запись $2^x \wedge 3$ не даст ничего, кроме сообщения 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 $x + 1/x \geq 2$ выполнено для всех $x > 0$.

В этом примере мы еще использовали знаки «строгих» неравенств (в Т_ЕХовских формулах они набираются непосредственно, как знаки $>$ и $<$) и нестрогих неравенств (знак «больше или равно» генерируется командой \geq , «меньше или равно» — командой \leq). Между прочим, если Вы употребите символы $<$ и $>$ в обычном тексте, вне формул, то вместо знаков «меньше» и «больше» увидите небольшой сюрприз.

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

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

$$\pi \approx 3,14 \qquad \text{\code{\pi}\code{\approx} 3{,}14}$$

Здесь команда `\pi` порождает греческую букву π , а команда `\approx` — знак \approx («приближенно равно»).

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

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

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

$$\frac{1}{2} + \frac{x}{2} = \frac{1+x}{2} \qquad \text{\code{\frac{1}{2}+\frac{x}{2}=\frac{1+x}{2}}}$$

3.4. Скобки

Круглые и квадратные скобки набираются просто так, для фигурных скобок используются команды `\{` и `\}`, для других также есть специальные команды, например `\langle` («левая угловая скобка» \langle).

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

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

Подробнее о скобках, размер которых выбирается автоматически, рассказано в следующей главе (раздел 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 палочки, обозначающие знак модуля, набираются непосредственно.

3.6. Штрихи и многоточия

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

Согласно формуле Лейбница,

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

Точнее говоря, формула Лейбница позволяет найти производную любого порядка от произведения двух функций.

Согласно формуле Лейбница,

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

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

Точнее говоря, формула Лейбница позволяет найти производную любого порядка от произведения двух функций.

Обратите, кстати, внимание на то, что точку мы поставили в конце выключной формулы (если бы мы поставили ее после знаков `$$`, то с нее начался бы абзац, следующий после формулы, что нелепо).

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 множители (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 `~` после инициалов великого Гаусса мы поставили, чтобы фамилия не могла перенестись на другую строчку отдельно от инициалов (см. стр. 60). `TeX` позволяет использовать команду `\ldots` и в обычном тексте, вне математических формул, для знака многоточия (см. стр. 59).

3.7. Функции типа синус

Функции наподобие \sin , \log и т. п., имена которых надо набирать прямым шрифтом, набираются с помощью специальных команд (обычно одноименных с обозначениями соответствующих функций). Полный список таких команд приведен в разделе II.1.2.

Нетрудно видеть, что $\log_{1/16} 2 = -1/4$, а $\sin(\pi/6) = 1/2$.	Нетрудно видеть, что $\log_{1/16} 2 = -1/4$, а $\sin(\pi/6) = 1/2$.
---	---

Заметьте, что основание логарифма задается как нижний индекс.

В стандартный набор команд Т_ЕXа не входят команды для функций tg и ctg (в англоязычных странах эти функции принято обозначать tan и cot соответственно). Большой беды тут нет, поскольку эти недостающие команды легко определить самому (см. главу II); возможно, кроме того, что Вы получили Л^AT_EX вместе с «русифицирующим стилем», в котором необходимые определения команд уже сделаны.

4. Разбиение исходного файла на части

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

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

```
\input имя_файла,
```

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

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

```
\documentstyle[11pt]{report}
\hfuzz=1.5pt
\pagestyle{plain}
\begin{document}
\input ch1.tex
\input ch2.tex
\input ch3.tex
\input ch4.tex
\end{document}
```

Ради реализма мы в этом примере включили в преамбулу парочку команд, которые могли бы там появиться и в реальной ситуации. Первая из них означает, что при верстке абзацев строки могут выбиваться за правую границу текста, но не более, чем на полтора пункта, а вторая — что номера страниц будут печататься снизу и при этом колонтитулов не будет. Позже мы рассмотрим эти вещи подробнее.

Каждую команду `\input` следует располагать на отдельной строке, как в вышеприведенном примере. Если расширение файла, являющегося аргументом команды `\input`, не указано, то Т_ЭХ по умолчанию считает, что это расширение имеет вид `.tex`.

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 Т_ЭХа (если таковые будут).

Если Вы хотите, чтобы Т_ЭХ прочитал только часть Вашего файла, можно воспользоваться командой `\endinput`. Если она присутствует в файле, читаемом Т_ЭХом с помощью команды `\input`, то файл будет прочитан только до строчки, в которой написано `\endinput`, после чего его чтение прекратится.

5. Обработка ошибок

В исходных текстах для Т_ЭХа, которые Вы будете готовить, неизбежно будут присутствовать ошибки. В настоящем разделе мы обсудим, как Т_ЭХ на них реагирует и как Вам, в свою очередь, следует реагировать на эти реакции Т_ЭХа.

Все сообщения, которые Т_ЭХ 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 Т_ЭХом файл, и расширение `.log`, поэтому на жаргоне протокол трансляции называется `log`-файлом. Когда трансляция 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 нестандартных ситуациях при верстке абзаца), при выдаче которых трансляция не прерывается (в разделе III.6 будет подробно рассказано, что означают такие предупреждения). В случае, однако, если Т_ЭХ наткнется на синтаксическую ошибку в исходном тексте, трансляция приостанавливается, а на экран выдается сообщение об ошибке.

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

```
\documentstyle{article}
\begin{document}
Чтобы овладеть \LaTeXом, надо потренироваться.
Следующая строка будет центрирована:
\begin{center}
Строка в центре.
\end{centrr}
А теперь попробуем формулы, например, такие,
как  $3^3=27$ . И еще выключную формулу:

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

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


Теперь обработайте наш файл `test.tex` с помощью Л^AT_EX. Вскоре Вы увидите на экране вот что:

```
! Undefined control sequence.
1.3 Чтобы овладеть \LaTeXом
      , надо потренироваться.
?
```

Первая строка Т_EXовского сообщения об ошибке всегда начинается с восклицательного знака, после которого идет краткое указание на характер ошибки (в нашем случае речь идет о том, что обнаружена несуществующая команда). Второй обязательный элемент сообщения об ошибке — строка, начинающаяся с 1., после которого идет номер строки исходного текста с ошибкой (в нашем случае 3). После номера на экран выдается сама эта строка, или та ее часть, которую Т_EX успел прочесть к моменту обнаружения ошибки. В нашем случае текст был прочитан до несуществующей команды «\LaTeXом» включительно (эта «команда» получилась потому, что мы забыли оставить пробел, ограничивающий имя команды \LaTeX — см. стр. 12), на которой Т_EX 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, на худой конец всегда можно нажать клавишу `x` или `X` (латинскую) и после этого «ввод»: тогда трансляция немедленно завершится. Делать так сразу же, однако, не очень разумно: в тексте вполне могут быть и другие ошибки, и за один сеанс хочется обнаружить их побольше. Поэтому лучше просто нажать клавишу «ввод»: при этом Т_EX 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 «\LaTeXом» (так что из печатного текста будет неясно, чем можно овладеть после тренировки). Нажимать «ввод» в ответ на сообщение об ошибке — наиболее распространенная на практике реакция: в 90% случаев этого вполне достаточно, и на первых порах можно этим и ограничиться. 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 `R` или `r`, а затем на «ввод»; при обнаружении дальнейших ошибок трансляция прерываться не будет (Т_EX будет обрабатывать ошибки, так, как если бы Вы все время нажимали на «ввод»), по экрану пронесутся сообщения об ошибках, а затем Вы сможете их спокойно изучить, просмотрев `log`-файл.

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

```
LaTeX error. See LaTeX manual for explanation.
      Type H <return> for immediate help.
! \begin{center} ended by \end{centrr}.
\@latexerr ...diate help.}\errmessage {#1}

\@checkend ...urrenvir \else \@badend {#1}
      \fi
\end ...me end#1\endcsname \@checkend {#1}
      \expandafter \endgroup \if@...
1.7 \end{centrr}
```

?

Это сообщение об ошибке начинается со слов `LaTeX error`. Такого рода сообщения не встроены в `TeX`, а создаются `LaTeX`ом (так что можно создать «русифицирующий стиль» для `LaTeX`а, при пользовании которым эти сообщения будут выдаваться по-русски). В нем, однако, по-прежнему присутствуют три основных элемента сообщений об ошибке: строка, начинающаяся с `!`, строка, начинающаяся с `l.`, и приглашение — вопросительный знак (на все остальное в этом сообщении внимания не обращайтесь — это интересно только `TeX`никам). Присутствует на экране и объяснение ошибки: из-за опечатки (`centrr` вместо `center`) получилось, что команда `\begin`, открывающая окружение, не соответствует команде `\end`, закрывающей его (см. раздел 2.8: 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>
```

1.9 как $(2x+1)^{\wedge}$

$$3=5x\$$$
. И еще выключную формулу:

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

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

```
! Undefined control sequence.
1.10 $$\frac{25}{36}=\lrfst
      (\frac{1}
```

?

Нажмем очередной раз на «ввод», и немедленно увидим сообщение еще об одной ошибке:

```
! Extra \right.
1.11 {1+\frac{1}{5}}\right)
      ^2
```

?

Откуда такое сообщение, если в строке 11 у нас все правильно? Оказывается, эта ошибка была «наведена» предыдущей! В самом деле, перед этим `TeX` проигнорировал «команду» `\lrfst`, набранную вместо `\left` (именно так `TeX` и делает, если в ответ на ошибку «несуществующая команда» нажать на клавишу «ввод»), так что команду `\left` `TeX` вообще «не видел»; теперь выходит так, что в тексте, который видит `TeX`, присутствует команда `\right` без команды `\left`, что запрещено (см. раздел 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_ЕX опять по-свойски исправит ошибку, и вскоре Вы увидите последнее сообщение об ошибке:

```
! Missing } inserted.
<inserted text>
      }
<to be read again>
      $
```

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

?

На сей раз ошибка в том, что мы забыли закрывающую фигурную скобку. Нажмем на «ввод»; Т_ЕX вставит недостающую скобку (в результате чего на печати получится забавная «формула» $\sqrt{4} = 2$, соответствующая исходному 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_ЕXа), после чего трансляция наконец завершится. Кстати, цифра 1 в квадратных скобках, появляющаяся при этом на экране, означает, что Т_ЕX сверстал страницу номер 1 и записал ее содержимое в dvi-файл. Теперь можно и посмотреть, как будет выглядеть наш текст на печати; из-за многочисленных ошибок вид будет несколько странный.

Количество различных сообщений об ошибках, которые может выдавать Т_ЕX, составляет несколько сотен, и нормальная реакция на них обычно такая же, как в нашем эксперименте, принципиальных отличий Вы не увидите (если Вы не Т_ЕXник, конечно); сейчас мы рассмотрим еще две типичные ошибки, реакция на которые должна быть иной.

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

```
! I can't find file 'ttst.tex'.
1.16 \input ttst
```

Please type another input file name:

В ответ на это следует набрать правильное имя файла и нажать на «ввод», и трансляция благополучно продолжится.³ Простое нажатие на «ввод» тут ничего не даст. Если команда `\input` с именем несуществующего файла попадется Т_ЕXу после того, как Вы в ответ на какую-то из прежних ошибок сказали R, то трансляция на этом месте прервется.

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 Л_АT_ЕX для обработки этого файла. Нажав сколько-то раз «ввод», мы обнаружим, что работа Т_ЕXа не закончилась, а на экран выдана звездочка: *. Эта звездочка — приглашение Т_ЕXа ввести еще текст или команды; она появляется, когда в исходном тексте отсутствует команда для Т_ЕXа «завершить работу» (в Л_АT_ЕXе эта команда входит в качестве составной

³Если вообще никакого файла нет (например, Т_ЕX запущен по ошибке), наберите `null` — это всегда существующий пустой файл. (В некоторых версиях — `nul` с одним l).

части в комплекс действий, выполняемых командой `\end{document}`). Теперь можно вводить с клавиатуры любой текст и команды — Т_ЭХ 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 Т_ЭХ «не заметит» команды `\end{document}`, и тогда Вы окажетесь лицом к лицу с Т_ЭХовским приглашением-звездочкой.

Наряду с пассивной реакцией на ошибки — все время нажимать на «ввод» или сказать R — есть и другая возможность: прямо с клавиатуры вносить исправления в тот текст, который «видит» Т_ЭХ. 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 Т_ЭХа для отладки.

Для того, 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 Т_ЭХом. Чтобы продемонстрировать это на практике, давайте приведем файл `test.tex` в исходное состояние, вернув в него последнюю строчку `\end{document}`, и еще раз запустим Л^AТ_ЭХ для его обработки. В ответ на первое же сообщение (по поводу несуществующей команды `\LaTeXom`) нажмем `i`, а затем, в ответ на приглашение `insert>`, наберем правильный текст

```
\LaTeX om
```

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 знаки в исходном тексте, между которыми должен был стоять пропущенный знак доллара, уже поглощены Т_ЭХом, 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 вообще не будет (ведь на сей раз в тексте, который видит Т_ЭХ, команда `\left` присутствует, а поэтому и на команду `\right` Т_ЭХ 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 посмотреть, как на сей раз будет выглядеть на печати наш текст; некоторые несуразности наподобие $\sqrt{4} = 2$ в нем останутся, но их будет меньше, чем если бы мы нажимали на «ввод»: не будет потеряно слово «L^AT_ЭХом», центрированная строка будет действительно центрирована, формула

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

будет выглядеть так, как надо. Это существенно, поскольку чем ближе к задуманному получится dvi-файл, тем меньше времени мы потратим в дальнейшем на выявление полиграфических недостатков и борьбу с ними (по поводу борьбы с полиграфическими недостатками см.

раздел III.6). Теперь остается внести исправления в исходный файл (справляясь с тем, что записано в log-файле) и запустить Л^AT_EX 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 T_EX прерывает трансляцию «по своей инициативе». На практике важны два случая:

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

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

и трансляция прекращается;

- когда T_EXу не хватило памяти — выдается сообщение наподобие такого:

```
! TeX capacity exceeded, sorry [main memory size=65533].
```

Нехватка памяти может случиться в результате некоторых ошибок, из-за которых T_EX «зацикливается»; тогда достаточно исправить ошибку. Иногда памяти T_EXу может действительно не хватить. Так бывает, если в тексте встречаются чудовищно длинные абзацы (длиннее, чем на 4–5 страниц) или сверхсложные таблицы с очень большим количеством строк и столбцов (см. главу VI по поводу верстки таблиц). Если Вы встретились с такой проблемой, то можно проконсультироваться со специалистом (или самому изучить по книге [2]), как использовать T_EX более эффективно (в частности, T_EX *можно* научить переваривать сколь угодно длинные абзацы). Можно также попробовать найти транслятор T_EXа, дающий возможность работать с увеличенным объемом памяти. Возможно, при этом Вам понадобится и более мощный компьютер. . .

Скажем пару слов про более экзотические способы реакции на ошибки.

Во-первых, в ответ на приглашение ? можно набрать h или H и нажать «ввод». В этом случае T_EX выдаст на экран дополнительную информацию по поводу Вашей ошибки (вряд ли Вы много из нее почерпнете, если Вы не T_EXник), а затем еще раз приглашение ? . Во-вторых, можно набрать s или S (и «ввод», естественно); результат будет такой же, как если бы Вы сказали R , с той разницей, что в случае, если аргументом команды \input служит несуществующий файл, трансляция не прервется, а будет выдано приглашение ввести верное имя файла. Наконец, можно набрать Q или q (и «ввод»): результат будет такой же, как от R, с той разницей, что на экран не будет выдаваться вообще ничего (в log-файл все будет записано).

Наконец, режимы реакции на ошибки, задаваемые с клавиатуры с помощью клавиш S, R или Q, можно также задать прямо в файле; для этого достаточно в преамбуле дать одну из перечисленных ниже команд.

Команда \scrollmode равносильна нажатию S

Команда \nonstopmode равносильна нажатию R

Команда \batchmode равносильна нажатию Q

6. Как читать дальше?

Наш обзор основных понятий завершен, и Вы уже можете подготовить с помощью Л^AT_EXа несложный текст. Дальнейшее чтение, в зависимости от Ваших потребностей, можно построить по-разному: несколько последующих глав почти независимы друг от друга.

В главе II рассказано про многочисленные тонкости и дополнительные средства, связанные с набором математических формул. Если Вас это не очень интересует, можете при первом чтении обращаться к ней только за справками. Знание материала этой главы не понадобится до последних разделов главы VI.

Глава III посвящена набору текста «в малом»: в ней рассказывается, в частности, как задавать в тексте шрифты разных форм 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 стили и чем они отличаются друг от друга, как устроить разбиение текста на разделы таким образом, чтобы L^AT_EX автоматически создавал заголовки этих разделов и к тому же автоматически их нумеровал, как оформлять титульный лист, как создать оглавление, и тому подобное. Для чтения этой главы подробное знание предыдущей не является обязательным.

В главе V описаны так называемые псевдорисунки — примитивные картинки, которые можно создавать, не выходя за рамки L^AT_EXa.

Последующие главы посвящены более сложным вопросам. В главе VI рассказывается о верстке таблиц с помощью L^AT_EXa. В главе VII объяснено, как можно повысить эффективность своей работы с L^AT_EXом, создавая собственные команды. Первые разделы этой главы можно читать параллельно с главой II. В главе VIII рассказывается о таких фундаментальных понятиях T_EXa, как «блоки» и «клей»; когда Вы усвоите материал этой главы, Вы сможете, в частности, создавать некоторые специальные эффекты при верстке более простыми средствами, чем это позволяет собственно L^AT_EX. Наконец, заключительная глава IX, предполагающая знание всего предыдущего материала, рассказывает, как можно изменить стандартный стиль оформления документов, предоставляемый нам L^AT_EXом, применительно к своим нуждам.

Глава II.

Как набирать формулы

1. Таблицы спецзнаков с комментариями

В этом разделе мы перечислим все математические знаки, предоставляемые ЛАТ_EХом. Знаков этих очень много, поэтому мы разобьем их на несколько групп.

1.1. Операции, отношения и просто значки

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

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

Имя команды, задающей прописную греческую букву, пишется с прописной буквы (например, буква Ψ задается командой `\Psi`). Некоторые прописные греческие буквы («альфа», например) совпадают по начертанию с латинскими, и для них специальных команд нет — надо просто набрать соответствующую латинскую букву прямым шрифтом (см. стр. 37 по поводу того, как это сделать). Не надо использовать греческие буквы Σ и Π из этой таблицы в качестве знаков суммы и произведения: для этих целей есть специальные команды, о которых пойдет речь дальше. Итак, вот прописные греческие буквы, не совпадающие по начертанию с латинскими:

Γ	<code>\Gamma</code>	Δ	<code>\Delta</code>	Θ	<code>\Theta</code>
----------	---------------------	----------	---------------------	----------	---------------------

Λ	<code>\Lambda</code>	Ξ	<code>\Xi</code>	Π	<code>\Pi</code>
Σ	<code>\Sigma</code>	Υ	<code>\Upsilon</code>	Φ	<code>\Phi</code>
Ψ	<code>\Psi</code>	Ω	<code>\Omega</code>		

Как видите, прописные греческие буквы печатаются, в отличие от строчных, прямым шрифтом. Если Вам нужны наклонные греческие буквы (вроде Σ), прочтите в разделе 2.3 о том, как их получить.

Следующая серия символов — символы, рассматриваемые Т_ЕXом как символы бинарных операций (наподобие знаков сложения, умножения и т. п.). Т_ЕX 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 полный 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>	\bigcirc	<code>\bigcirc</code>	\triangleup	<code>\triangleup</code>
∇	<code>\nabla</code>				

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

В следующей таблице мы собрали символы «бинарных отношений». Вокруг них Т_ЕX также оставляет дополнительные пробелы (не такие, как вокруг символов бинарных операций). Вообще говоря, нет смысла много задумываться об этих пробелах, поскольку Т_ЕX 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. Команда `\mid` в этой таблице определяет вертикальную черточку, рассматриваемую как знак бинарного отношения; ее *не* следует употреблять, если вертикальная черточка употребляется как аналог скобки (например, как знак абсолютной величины).

$<$	<code><</code>	$>$	<code>></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>	\sqsubset	<code>\sqsubset</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>	\Join	<code>\Join</code>
\propto	<code>\propto</code>				

В следующей таблице собраны стрелки различных видов. Среди них есть и вертикальные; как их использовать при наборе формул, Вы узнаете попозже, в разделе 3.2.

\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>	\leadsto	<code>\leadsto</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>	\rightharpoonup	<code>\rightharpoonup</code>	\leftharpoondown	<code>\leftharpoondown</code>
\rightharpoondown	<code>\rightharpoondown</code>	\rightleftharpoons	<code>\rightleftharpoons</code>		

Из привычных российскому читателю символов в вышеприведенных таблицах нет знаков \leq и \geq , более привычных, чем \geq и \leq ; кроме того, греческая буква «каппа» лучше смотрится в виде κ , чем в виде κ , получающемся в результате действия команды `\kappa`. Эти символы отсутствуют в стандартных \TeX овском и \LaTeX овском наборах; при пользовании $AMS-\LaTeX$ ом они становятся доступными.

1.2. Операции с пределами и без

В следующей таблице собраны «операции типа синус» — команды для печати математических операций типа \sin , \log и т. п., обозначаемых последовательностью букв, набираемых прямым шрифтом. К любой из этих команд можно поставить верхний и/или нижний индекс (см. пример на стр. 22).

\log	<code>\log</code>	\lg	<code>\lg</code>	\ln	<code>\ln</code>
\arg	<code>\arg</code>	\ker	<code>\ker</code>	\dim	<code>\dim</code>
\hom	<code>\hom</code>	\deg	<code>\deg</code>	\exp	<code>\exp</code>
\sin	<code>\sin</code>	\arcsin	<code>\arcsin</code>	\cos	<code>\cos</code>
\arccos	<code>\arccos</code>	\tan	<code>\tan</code>	\arctan	<code>\arctan</code>
\cot	<code>\cot</code>	\sec	<code>\sec</code>	\csc	<code>\csc</code>
\sinh	<code>\sinh</code>	\cosh	<code>\cosh</code>	\tanh	<code>\tanh</code>
\coth	<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 русской литературе, однако же, принято обозначать тангенс tg . Так как в стандартном комплекте \TeX а или \LaTeX а команды для этого нет, ее приходится, при необходимости, делать самому. Это просто: в преамбуле документа надо написать

`\newcommand{\tg}{\mathop{\rm 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 математического оператора (скажем, `arctg`), которое должно появиться на печати. Если Вы получили \TeX вместе с русификацией, то не исключено, что в ней уже определены команды для принятых в России обозначений тангенса, арктангенса и т. п. (по крайней мере они есть в той русификации, которая использовалась при верстке этой книги).

Описанный выше способ определения команд для «операций типа синус» является частным случаем существующей в \TeX конструкции для определения новых команд (см. главу VII).

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

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

Обратите внимание, что скобки вокруг `mod 11` получаются автоматически; правая часть сравнения — весь текст, заключенный между `\equiv` и `\pmod`. Иногда, кроме того, символ `mod` используется как символ бинарной операции, например, так:

$$f_*(x) = f(x) \bmod G \qquad \text{\code{f_*(x)=f(x)\bmod G}}$$

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

```


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


```

В этом примере существенно, что формула была выключной; во внутритекстовой формуле «пределы» печатаются на тех же местах, что и индексы (можно добиться, чтоб они и во внутритекстовой формуле были сверху и снизу — см. ниже).

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

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

Вот список операций, ведущих себя так же, как `\sum`:

Σ	<code>\sum</code>	\prod	<code>\prod</code>	\cup	<code>\bigcup</code>
\cap	<code>\bigcap</code>	\coprod	<code>\coprod</code>	\oplus	<code>\bigoplus</code>
\otimes	<code>\bigotimes</code>	\odot	<code>\bigodot</code>	\vee	<code>\bigvee</code>
\wedge	<code>\bigwedge</code>	\uplus	<code>\biguplus</code>	\sqcup	<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>				

Еще одна «математическая операция», для которой требуются «пределы», — это интеграл. В ЛАТЭХе есть команды `\int` для обычного знака интеграла \int и `\oint` для знака «контурного интеграла» \oint . При этом, для экономии места, пределы интегрирования печатаются не сверху и снизу от знаков интеграла, а по бокам (даже и в выключных формулах):

$$\int_0^1 x^2 dx = 1/6 \qquad \begin{array}{l} \$\$ \\ \int_0^1 x^2 dx = 1/6 \\ \$\$ \end{array}$$

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:

$$\int_0^1 x^2 dx = 1/6 \qquad \begin{array}{l} \$\$ \\ \int \limits_0^1 x^2 dx = 1/6 \\ \$\$ \end{array}$$

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

Если, с другой стороны, надо, чтобы «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»:

$$\prod_{i=1}^n i = n! \qquad \begin{array}{l} \$\$ \\ \prod \nolimits_{i=1}^n i = n! \\ \$\$ \end{array}$$

1.3. Разное

Мы уже перечислили почти все символы, используемые ЛАТЭХом в математических формулах. Остались скобки различных видов (им будет посвящен специальный раздел 2.5), а также ряд значков (среди них есть и часто встречающихся), не входящих ни в какой из разделов нашей классификации. Они собраны в следующей таблице.

∂	<code>\partial</code>	\triangle	<code>\triangle</code>	\angle	<code>\angle</code>
∞	<code>\infty</code>	\forall	<code>\forall</code>	\exists	<code>\exists</code>
\emptyset	<code>\emptyset</code>	\neg	<code>\neg</code>	\aleph	<code>\aleph</code>
\prime	<code>\prime</code>	\hbar	<code>\hbar</code>	∇	<code>\nabla</code>

i	<code>\imath</code>	j	<code>\jmath</code>	ℓ	<code>\ell</code>
$\sqrt{\quad}$	<code>\surd</code>	\flat	<code>\flat</code>	\sharp	<code>\sharp</code>
\natural	<code>\natural</code>	\top	<code>\top</code>	\perp	<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>
\mho	<code>\mho</code>	\Box	<code>\Box</code>	\Diamond	<code>\Diamond</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.

Символы, обозначаемые командами `\imath` и `\jmath`, нужны для постановки дополнительных значков над буквами i и j (об этом пойдет речь в разделе 2.7).

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

Наша последняя таблица — таблица синонимов. Некоторые математические символы можно набирать двумя различными способами; соответствующие варианты собраны в следующей таблице.

$*$	<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>
\parallel	<code>\Vert</code> или <code>\parallel</code>

2. Важные мелочи

2.1. Нумерация формул

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

Выключная формула, которую Вы нумеруете, должна быть оформлена как окружение `equation` (знаков $$$$ быть не должно!). Каждая такая формула на печати автоматически получит но-

мер. Чтобы на него можно было ссылаться, надо формулу пометить: в любом месте между `\begin{equation}` и `\end{equation}` поставить команду `\label`, и после этого команда `\ref` будет генерировать номер формулы (см. стр. 17). Поясним все сказанное примером:

<p>Каждый первоклассник должен знать, что</p> $7 \times 9 = 63. \quad (1)$ <p>.....</p> <p>Из формулы (1) следует, что $63/9 = 7$.</p>	<p>Каждый первоклассник должен знать, что</p> <pre>\begin{equation} \label{trivial} 7\times9=63. \end{equation}</pre> <p>.....</p> <p>Из формулы~(\ref{trivial}) следует, что $63/9=7$.</p>
---	--

Знак ~ мы поставили, чтобы номер формулы и слово «формула» не попали на разные строки (см. стр. 60). Обратите внимание, что скобки вокруг номера формулы, сгенерированного командой `\ref`, автоматически *не* ставятся.

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 стиля документа (см. стр. 12, 97): например, в стиле `article` (статья) формулы имеют сплошную нумерацию, а в стиле `book` (книга) нумерация формул начинается заново в каждой главе, и номер, скажем, формулы 5 из главы 3, генерируемый окружением `equation`, имеет вид (3.5). В разделе, посвященном модификации стандартных стилей, мы расскажем, как можно самостоятельно менять вид номеров формул.

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

<p>Простое тождество</p> $7 \times 9 = 63 \quad (3.2)$ <p>известно каждому первокласснику.</p>	<p>Простое тождество</p> <pre>\$\$ 7\times9=63\eqno (3.2) \$\$</pre> <p>известно каждому первокласснику.</p>
--	--

Выключная формула, нумеруемая с помощью команды `\eqno`, должна быть оформлена с помощью знаков `$$`; номером формулы будет служить весь текст, заключенный между `\eqno` и закрывающими формулу `$$`; этот текст обрабатывается Т_ЭХом 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:

<p>Менее простое тождество</p> <p>(*) $\sin^2 x + \cos^2 x = 1$</p> <p>известно каждому десятикласснику.</p>	<p>Менее простое тождество</p> <pre>\$\$ \sin^2x+\cos^2x=1 \leqno (*) \$\$</pre> <p>известно каждому десятикласснику.</p>
---	---

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

2.2. Переносы в формулах

При необходимости \TeX может перенести часть внутритекстовой формулы на другую строку. Такие переносы возможны после знаков «бинарных отношений», наподобие знака равенства (см. стр. 31) или «бинарных операций», наподобие знаков сложения или умножения (см. стр. 31), причем последний знак в строке, вопреки российской традиции, *не* дублируется в начале следующей. Если это Вас раздражает, или Вы вообще не хотите, чтобы внутритекстовые формулы переносились, можно воспользоваться тем обстоятельством, что \TeX не разрывает при переносе часть формулы, заключенную в фигурные скобки. В частности, можно заключить в фигурные скобки *всю* формулу, от открывающего ее знака доллара до закрывающего: после этого можете быть уверены, что переноса этой формулы ни при каких обстоятельствах не произойдет.

Выключные формулы, в отличие от внутритекстовых, \TeX *никогда* не переносит. Если выключная формула не помещается в строку, то при трансляции Вы получите сообщение «`Overfull \hbox`» (в разделе 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 разделе 3.3.

2.3. Некурсивные шрифты в математической формуле

Как уже говорилось, по умолчанию все латинские буквы в формулах набираются курсивным шрифтом. Если требуется иной шрифт, надо дать команду переключения на него. В разделе I.2.5 приведено несколько команд переключения шрифта; их полный список содержится в разделе III.4 (используйте в формулах только команды, меняющие гарнитуру, наподобие `\bf` или `\tt`; для ручного управления размерами символов в формулах предназначены команды, описанные в разделе 4.2). Чаще всего используются такие команды, как `\rm` для переключения на прямой шрифт и `\bf` для переключения на полужирный шрифт. Если команда переключения шрифта дана внутри группы, то после выхода из группы шрифт, как обычно, восстанавливается:

Обозначение \mathbf{P}^n используется для n -мерного проективного пространства.

Обозначение $\{\mathbf{P}\}^n$ используется для n -мерного проективного пространства.

Команда `\mit` возвращает шрифт к «математическому курсиву», принятому по умолчанию; нужда в ней возникает редко. Один случай, когда эта команда реально полезна, — это если мы хотим получить прописные греческие буквы, напечатанные наклонным шрифтом. В этом случае надо дать (внутри группы) команду `\mit` и команду для соответствующей греческой буквы:

$$\Sigma_a^X = C$$

$$\{\mit\Sigma\}^X_{a=C}$$

Наряду со шрифтами, применяемыми в текстах, Л^AT_EX предоставляет еще «каллиграфический» шрифт, который можно употреблять только в формулах. Этим шрифтом можно печатать только прописные латинские буквы; переключение на него задается командой `\cal`. Вот пример:

Касательное расслоение к многообразию X обозначается или так: \mathcal{T}_X , или так: T_X .

Касательное расслоение к многообразию X обозначается или так: `\cal T_X`, или так: `T_X`.

Символ `~` в этих примерах — это «символ неразрывного пробела» (см. стр. 60); мы поставили его, чтоб строчка не начиналась с формулы.

Все перечисленные команды смены шрифта в формулах действуют только на латинские буквы и не оказывают влияния на начертание греческих букв и спецсимволов.

Можно сделать так, чтобы по умолчанию латинские буквы в формулах набирались не курсивом, а полужирным прямым шрифтом. Для этого надо в преамбуле документа поставить команду `\boldmath`.

Если Вы хотите включить в формулу какой-либо текст, то одной команды `\rm` для этого отнюдь не достаточно: любой текст, пусть даже он набирается прямым шрифтом, T_EX рассматривает как часть математической формулы, и в соответствии с этим игнорирует те пробелы, которые ставите Вы, и расставляет пробелы по собственным правилам:

$$\sqrt{x^3} = x \text{ for all } x.$$

$$\sqrt{x^3}=x \text{ \rm for all } x.$$

Что на самом деле надо сделать, чтобы вставить текст в формулу, описано в следующем разделе.

2.4. Включение текста в формулы

В математическую формулу можно включить фрагмент обычного текста с помощью Л^AT_EXовской команды `\mbox`. В следующем примере продемонстрировано, как это можно сделать; в нем используется еще команда `\qqquad`, делающая в тексте или формуле пробел размером `2em` (см. стр. 16 по поводу единиц длины, применяемых T_EXом); подробнее по поводу команд, создающих пробелы в формулах, см. раздел 4.1; по поводу команд, создающих пробелы в тексте, см. раздел III.2.3.

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

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

Аргумент команды `\mbox` обрабатывается T_EXом как обычный текст: пробелы не игнорируются, слова набираются не математическим курсивом, а тем же шрифтом, который был текущим перед началом формулы (у нас это был обычный прямой шрифт; если Вы хотите, чтобы шрифт был другой, можно внутри аргумента команды `\mbox` дать команду смены шрифта).

Весь текст, являющийся аргументом команды `\mbox`, будет напечатан в одну строку. В приведенном примере мы оставили пробел перед закрывающей фигурной скобкой, чтобы обеспечить пробел между текстом и формулой (фрагмент текста, созданный командой `\mbox`, рассматривается Т_ЭХом как одна большая буква; пробел в формуле между «буквой», содержащей текст из `\mbox`, и буквой x будет недостаточен). Команда `\qqquad` была использована по аналогичной причине.

На самом деле можно было бы написать даже так:

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

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

При включении в текста в формулы с помощью команды `\mbox` важно иметь в виду вот что. Как читатель, видимо, уже заметил, в математических формулах верхние и нижние индексы, числитель и знаменатель дробей, созданных с помощью команды `\frac`, и тому подобные фрагменты набираются более мелким шрифтом, чем остальная часть формулы. Однако в тексте, созданном с помощью `\mbox`, размер шрифта не изменится, в какую бы часть формулы этот текст не попал.

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

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 больше размером, чем обычные. В Т_ЭХе 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 эти на печати размер этих скобок будет соответствовать высоте фрагмента формулы, заключенного между `\left` и `\right`.

Конструкция с `\left` и `\right` применима не только к круглым скобкам. В нижеследующей таблице перечислены скобки и некоторые другие символы, которые с помощью `\left`

и `\right` автоматически принимают нужный размер. Т_ЕXнический термин для таких символов — ограничители (по-английски *delimiters*).

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

Вместо `\left\langle` можно писать `\left<`, и аналогичным образом вместо `\right\rangle` можно писать `\right>` (однако же `<` — это не `\langle`!). Кроме знаков, перечисленных в этой таблице, менять свои размеры под действием `\left` и `\right` могут и вертикальные стрелки из таблицы на стр. 32.

Вместе с каждой командой `\left` в формуле должна присутствовать соответствующая ей команда `\right`, в противном случае Т_ЕX выдаст сообщение об ошибке. Вместе с тем Т_ЕX 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 Т_ЕX 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):

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

```


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


```

В этом примере используется пока неизвестная Вам команда `\,`, создающая дополнительный маленький пробел между $f(x)$ и dx — это один из немногих случаев, когда Т_ЕX не может автоматически создать требуемые пробелы, и ему надо помочь. Подробно о таких вещах речь пойдет ниже, в разделе 4.1. Другой пример использования ограничителя без пары таков:

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

```


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


```

Наконец, важный пример использования ограничителей без пары — использование их для верстки систем уравнений, о чем пойдет речь в разделе 3.2.

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

$$||x + 1| - |x - 1|| \quad \text{\left| |x+1|-|x-1|\right|}$$

Для ясности хотелось бы, чтобы внешние знаки модуля были повыше, чем внутренние, но этого не получается: поскольку вся формула лишней высотой не обладает, то и команды `\left` и `\right` не считают нужным увеличить ограничители, в которые формула заключена.

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

$$\left(\sum_{k=1}^n x^k \right)^2 \quad \begin{array}{l} \text{\$} \\ \text{\left(} \\ \text{\sum_{k=1}^n x^k} \\ \text{\right)^2} \\ \text{\$} \end{array}$$

Во всех этих случаях имеет смысл указать размер ограничителя явно. Для этого предусмотрены \TeX овские команды `\bigl`, `\Bigl`, `\biggl` и `\Biggl` для левых ограничителей и `\bigr`, `\Bigr`, `\biggr` и `\Biggr` для правых ограничителей. Мы перечислили эти команды в порядке возрастания размера создаваемого ими ограничителя. В частности, вышеприведенные примеры выглядели бы лучше, если бы мы в примере со знаками модуля написали так:

$$\Bigl| |x + 1| - |x - 1| \Bigr| \quad \text{\Bigl| |x+1|-|x-1|\Bigr|}$$

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

$$\Bigl(\sum_{k=1}^n x^k \Bigr)^2 \quad \begin{array}{l} \text{\$} \\ \text{\Bigl(} \\ \text{\sum_{k=1}^n x^k} \\ \text{\Bigr)^2} \\ \text{\$} \end{array}$$

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

К сожалению, команды для явного указания размера ограничителя имеют, при использовании их в \LaTeX е, одну неприятную особенность: если «основной шрифт» документа крупнее, чем кегль 10 (иными словами, если указаны стилевые опции `11pt` или `12pt` — см. стр. 97 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». Поэтому при необходимости явного указания размеров ограничителей конкретную команду для выбора размера приходится иной раз подбирать экспериментально.

2.6. Перечеркнутые символы

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

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

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

Кстати, для получения знака «не принадлежит» лучше писать `\notin`, а не `\not\in`, — при этом знак получится более красивым.

2.7. Надстрочные знаки

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

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

Согласно часто используемым обозначениям,

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

Особенно часто так пишут в научно-популярных книгах.

Согласно часто используемым обозначениям,

`$$`

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

`$$`

Особенно часто так пишут в научно-популярных книгах.

Для постановки других значков над буквами в формулах предусмотрены команды, перечисленные в нижеследующей 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}

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

Если поставить значок над буквой *i* или *j*, так, чтобы сохранилась и точка над буквой, то это будет некрасиво. Поэтому ставить акценты следует ставить не прямо над этими буквами, а над символами *i* и *j* (см. таблицу на стр. 35):

Писать \tilde{i} некрасиво; лучше писать так: \tilde{i} .

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

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

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

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

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

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

Рассмотрим вектор \overrightarrow{AB} .

Рассмотрим вектор
`\overrightarrow{AB}`.

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

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

TeX позволяет поставить надстрочные знаки над буквами не только в математической формуле, но и в обычном тексте (в этом случае такие знаки обычно называют «диакритическими»), но команды для постановки этих знаков совершенно другие. См. стр. 63.

2.8. Альтернативные обозначения для математических формул

Наряду со стандартными TeXовскими обозначениями для математических формул, L^AT_EX предоставляет альтернативные обозначения. Именно, внутритекстовую формулу, которая в стандартных обозначениях ограничивается одним знаком доллара в начале и одним в конце, можно вместо этого заключить в знаки `\(` (в начале) и `\)` (в конце). Другой вариант обозначений для внутритекстовой формулы, предоставляемый L^AT_EXом, — написать `\begin{math}` в начале формулы и `\end{math}` в конце (иными словами, внутритекстовая формула может быть оформлена как окружение с именем `math`):

$2 \times 2 = 4$, или $2 \times 2 = 4$ — одно
и то же.

`$2\times2=4$`, или
`\(2\times2=4\)`
 --- одно и то же.

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

Эти альтернативные обозначения полностью эквивалентны стандартным (со знаками доллара), за одним важным исключением: если выключные формулы обозначаются L^AT_EXовскими, а не TeXовскими обозначениями, то можно сделать так, что выключные формулы будут не центрированы, а прижаты влево (см. стр. 98).

Создатель L^AT_EXа Лесли Лампорт надеялся, что его обозначения для формул будут удобнее стандартных, поскольку в них различаются символы, «открывающие» и «закрывающие» формулу, что позволяет легче находить ошибки в исходном тексте. По мнению автора, эти надежды оправдались для коротких обозначений `\(`, `\)`, `\[` и `\]`, в то время как более длинные обозначения с помощью `\begin` и `\end` оказались слишком громоздкими. Аргумент в пользу стандартных обозначений — то обстоятельство, что при переводе файла из L^AT_EXа в plain TeX или AMS-TeX не приходится заменять на доллар каждый знак `\(` и так далее.

3. Одно над другим

В этом разделе речь пойдет о тех случаях, когда в формуле необходимо поместить один символ над другим. В разделе 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.

3.1. 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 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.

Чтобы расположить части формулы по варианту 1), удобно воспользоваться Т_ЕXовской командой `\atop`:

В старину вместо Γ_{ij}^k писали $\left\{ \begin{matrix} ij \\ k \end{matrix} \right\}$.	В старину вместо Γ_{ij}^k писали $\left\{ ij \atop k \right\}$.
--	---

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

Часто встречающийся случай, когда выражения должны быть расположены так, как это делает команда `\atop` — это «биномиальные коэффициенты»; их можно было бы набирать как в вышеприведенном примере, с помощью `\left(, \atop` и `\right)`, но быстрее воспользоваться готовой командой `\choose`, которую предоставляет нам Т_ЕX:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \begin{matrix} \$\$ \\ \{n\choose k\}=\frac{n!}{k!(n-k)!} \\ \$\$ \end{matrix}$$

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

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`.

Рассмотрим второй случай, когда нижняя часть формулы должна остаться на уровне строки. Чтобы добиться этого эффекта, используется Л_АT_ЕXовская 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`

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

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

получается следующим образом:

```


$$$
\underbrace{1+3+5+7+
\cdots+2n-1}_{\mbox{\$n\$ слагаемых}}=n^2
$$$


```

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

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

```


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


```

Разумеется, совсем не обязательно записывать формулу с такими отступами и пробелами, как в нашем примере; мы пытались расположить текст так, чтобы яснее была его структура, благо пробелы в формулах `TeX` все равно игнорирует.

Рассмотренные приемы не исчерпывают всех случаев, когда требуется расположить одну часть формулы над другой. Например, при печати матриц желательно, чтобы элементы матрицы, расположенные в одном столбце, и на бумаге были расположены точно один под другим. Добиться этого с помощью `\atop` практически невозможно. В следующем разделе мы опишем, как грамотно набирать матрицы, системы уравнений и т. п.

3.2. Набор матриц

Чтобы набрать с помощью `LaTeX` матрицу, надо воспользоваться окружением `array`. Прежде, чем описывать, как это окружение работает, разберем такой пример:

```


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


```

Посмотрим, как устроен исходный текст, давший на печати эту матрицу. Всякая матрица состоит из *строк* и *столбцов*; 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 `&`. Далее, после `\begin{array}`, открывающего окружение, должна следовать (в фигурных скобках, поскольку это параметр окружения `array`) так называемая *преамбула* матрицы, описывающая, сколько и каких столбцов должно быть в матрице. В нашем случае преамбула представляет собой четыре буквы `cccc`. Это значит, что в матрице 4 столбца (по букве на столбец), и что содержимое каждого из этих столбцов должно быть расположено по центру столбца (поскольку каждая из букв — буква `c`). Кроме `c`, в преамбуле может стоять буква `l`, означающая, что соответствующий столбец будет выровнен по левому краю, или `r`, означающая, что столбец будет выровнен по правому краю.

В третьей строке матрицы мы использовали вертикальное многоточие, генерируемое командой `\vdots`, и диагональное многоточие, генерируемое командой `\ddots`. Эти команды можно использовать не только внутри окружения `array`, но и в любом месте в математических формулах.

Интересно, что в стандартном ЛАТЭХовском наборе не предусмотрено команды для «диагонального многоточия» `\cdot\cdot`, направленного в противоположную по сравнению с `demoddots` сторону, хотя создать такую команду с помощью ТЭХовских средств ничуть не сложнее, чем команду `\ddots`.

Нашей матрице не хватает еще скобок; чтобы их создать, надо перед `\begin{array}` написать `\left(`, а после `\end{array}` написать `\right)` (см. раздел 2.5).

Исходный текст, генерирующий на печати нашу матрицу, расположен таким образом, чтобы одной строке исходного текста соответствовал одна строка матрицы. Такое расположение мы выбрали только для удобства чтения, но, вообще говоря, оно совершенно не обязательно: бывает, что на протяжении нескольких строк приходится тянуть текст, который пойдет в одну строку на печати, а иногда в одной строке исходного текста помещается несколько строк матрицы. Повторим еще раз, что в ЛАТЭХе разбиение текста на строки роли не играет: конец строки воспринимается просто как пробел.

Окружение `array` можно, разумеется, использовать не только для верстки матриц в математическом смысле этого слова: фактически это окружение создает таблицы, состоящие из строк и столбцов. Вот, например, как можно напечатать треугольник Паскаля:

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

Исходный текст для него выглядит так:

```

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

```

Как видите, если какая-то графа в нашей таблице должна быть пуста, то между (или перед, если эта графа — первая в своей строке) соответствующими знаками & нужно просто ничего не писать (или оставить сколько угодно пробелов). Если после того, что Вы написали в строке, до конца строки идут только пустые графы, то можно не дописывать до конца значки &, а сразу написать \\.

Разберем еще один типичный пример: верстку системы уравнений с помощью окружения `array`. Посмотрите, как получается такая система уравнений:

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

```

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

```

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

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. раздел 2.1). Если пометить какое-либо уравнение с помощью команды `\label`, то в дальнейшем можно на него ссылаться с помощью команды `\ref` (тогда автоматически напечатается номер уравнения) или `\pageref` (тогда автоматически напечатается номер страницы, на которую попало это уравнение). По поводу `\ref` и `\pageref` см. раздел I.2.11. Итак, пример:

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

```

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

```

Обратите внимание, что фигурной скобки, охватывающей систему уравнений, окружение `eqnarray` не создает. В этом примере символ `~` между «стр.» и `\pageref` поставлен, чтобы слово «стр.» и номер страницы не попали на разные строки (см. стр. 60); для аналогичных целей мы использовали этот символ и вторично.

При использовании окружения `eqnarray` не надо писать знаков `$$` (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 `\`):

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

$$\sqrt{576} = 24 \quad (4)$$

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

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

Давайте теперь подумаем, как с помощью окружения `array` можно набирать так называемые «коммутативные диаграммы» такого, например, вида:

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

Разумно реализовать эту диаграмму как таблицу с тремя строками и девятью столбцами (по столбцу на каждую горизонтальную стрелку и на каждый нуль или букву). Как создать буквы над горизонтальными стрелками, мы тоже уже говорили: с помощью `\stackrel` (см. стр. 44). Осталось понять, как получить буквы справа от вертикальных стрелок. Для экономии попробуем сначала нарисовать диаграмму с *одной* вертикальной стрелкой. Вот первая попытка:

$$\begin{array}{c} E \\ \downarrow q \\ F \end{array}$$

```

$$
\begin{array}{c}
E \\
E \\\ \downarrow q \\
F
\end{array}
$$
```

Вроде бы получилось, но не совсем то, что нужно: стрелка расположена не по центру. Не улучшит положения, если в преамбуле написать `r` вместо `c`; если написать `l`, то будет чуть лучше, но стрелка все равно будет не в центре. Чтобы всего этого избежать, надо вместо `q` написать `\leftteqn{q}`: после этого столбцы нашей таблицы будут отцентрированы так, словно во второй строке есть только стрелка (команда `\downarrow`), а буква `q` при этом центрировании принята во внимание не будет (но будет напечатана)! В результате получится именно то, чего мы и хотели.

Для интересующихся объясним механизм действия этого приема. В процессе верстки текста `TeX` учитывает только, сколько места надо оставить на каждую букву, но не учитывает, как именно эта буква будет выглядеть на печати и сколько места она реально будет занимать. В обычных условиях на каждый символ оставляется именно столько места, сколько он занимает на самом деле, однако в `TeX` предусмотрены и специальные команды, позволяющие отвести тексту меньше или больше места, чем он займет фактически. В частности, команда `\leftteqn` печатает формулу, являющуюся ее аргументом, но при этом сообщает `TeX`у, что по горизонтали эта формула не занимает места вообще. Стало быть, с точки зрения `TeX`а ширина элемента, стоящего во второй строке нашей таблицы, определяется только шириной стрелки, и при центрировании текст располагается так, чтобы именно стрелка была на равном расстоянии от краев, сколь бы длинна на самом деле не была формула — аргумент `\leftteqn`.

Теперь можно сделать и всю коммутативную диаграмму целиком. Держитесь:

Возможно, Вы сочтете более удачным такое расположение, при котором продолжение формулы расположено точно под правой частью формулы в первой строке. Чтобы достичь этого эффекта, можно воспользоваться окружением `array` с преамбулой `rcl` или, если нужна автоматическая нумерация, окружением `eqnarray`:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

```


$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$


```

Впрочем, если левая часть Вашей формулы занимает много места, то этот прием не работает: вторая строка начнется только под правой частью формулы в первой строке, что будет выглядеть некрасиво, да и на страницу эта вторая строка может не поместиться.

Наконец, вот еще один прием. Формулу

$$\int_0^x e^{-t^2} dt = x - \frac{x^3}{1! \cdot 3} + \frac{x^5}{2! \cdot 5} - \frac{x^7}{3! \cdot 7} + \dots + (-1)^n \frac{x^{2n+1}}{n! \cdot (2n+1)} + \dots$$

мы получили следующим образом:

```

\begin{eqnarray*}
\lefteqn{\int_0^x}
e^{-t^2} dt = x - \frac{x^3}{1! \cdot 3}
+ \frac{x^5}{2! \cdot 5} -
\frac{x^7}{3! \cdot 7} + \dots \\
& & + (-1)^n \frac{x^{2n+1}}{n! \cdot (2n+1)} + \dots
\end{eqnarray*}

```

Здесь мы опять воспользовались командой `\lefteqn`, о которой уже шла речь в предыдущем пункте в связи с набором «коммутативных диаграмм». См. выше по поводу `{}` перед знаком минуса во второй строке.

На сей раз команда `\lefteqn` проявляет себя так: поскольку под формулу, являющуюся ее аргументом, `TeX` места по горизонтали не отводит, то получается, что первый столбец нашей таблицы имеет нулевую ширину: в первой строчке в этом столбце расположен весьма длинный текст, но `TeX`, обманутый командой `\lefteqn`, считает, что места этот текст не занимает; во второй же строчке в этом столбце вообще ничего нет. Во втором столбце нашей таблицы также ничего нет (уже по-честному). Поэтому третий столбец начинается довольно близко от края: `LaTeX` отступает только на суммарную ширину первого и второго столбцов (она равна нулю) плюс те промежутки, которые `LaTeX` автоматически делает между столбцами в окружениях `array`, `eqnarray` или `eqnarray*`. Создатель `TeX`а Дональд Кнут назвал такого рода приемы работы с `TeX`ом «грязными трюками» (dirty tricks). Впрочем, при написании таких больших и сложных `TeX`овских макропакетов, как `LaTeX`, используются трюки и похлеще.

4. Тонкая настройка

В этом разделе мы рассмотрим некоторые более изысканные вопросы, связанные с набором математических формул.

4.1. Пробелы вручную

Бывают случаи, когда промежутки между символами в формулах, выбранные Т_ЕXом автоматически, выглядят неудачно. В этом случае в формулу можно включить команды, задающие промежутки в явном виде. Основные из этих команд таковы:

<code>\quad</code>	Пробел в 1em
<code>\qquad</code>	Пробел в 2em
<code>\,</code>	«Тонкий пробел»
<code>\:</code>	«Средний пробел»
<code>\;</code>	«Толстый пробел»
<code>\!</code>	«Отрицательный тонкий пробел»

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

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

Возьмем, например, $\int f(x) dx$
или $\iint f dx dy$, или $\sqrt{3}x$.

Возьмем, например,
`\int f(x)\,dx`
или `\int\!\!\int f\,dxdy`,
или `\sqrt{3}\,x`.

Команда `\qquad` полезна для отделения текста, входящего в формулу, от собственно формулы (см. стр. 38).

4.2. Размер символов в формулах

В большинстве случаев Вам не приходится задумываться о том, какой размер будут иметь символы в формуле: как Вы, вероятно, уже заметили, Т_ЕX 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 Т_ЕX выбирает размеры символов в формулах и как можно на него при этом влиять.

При наборе формулы Т_ЕX в каждый момент руководствуется одним из следующих «стилей» (эти стили не имеют ничего общего со стилями оформления документа, о которых говорилось в разделе I.2.4):

<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 «поскромнее» (например, в выключном стиле верхние индексы поднимаются повыше, а нижние — пониже, чем в текстовом; в текстовом стиле «пределы» операций записываются не сверху, а сбоку — ср. раздел I.2). В стиле для индексов используются более мелкие шрифты, чем в выключном или текстовом (а в стиле для индексов к индексам — еще более мелкие). Выбираются стили набора формул следующим образом: выключная

формула начинает набираться в выключном стиле, внутритекстовая — в текстовом стиле; далее, если в момент действия какого-то из стилей встретится команда `\frac` (или `\atop`), то для набора числителя и знаменателя \TeX переключается на следующий по порядку стиль из вышеприведенной таблицы; если в момент действия выключного или текстового стиля встретится верхний или нижний индекс (показатель степени мы также рассматриваем как верхний индекс — в математическое содержание формул \TeX не вникает), то этот индекс начинает набираться стилем для индексов; если индекс встретится в момент действия стиля для индексов или индексов к индексам, то набираться он будет в стиле «индексы к индексам». Например, при наборе формулы

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

\TeX использует выключной стиль при наборе $x +$, текстовый стиль при наборе $y^2 + 3 - z^{x^7 - y^7}$ и $1 + \cos^2 x$, стиль для индексов при наборе $x^7 - y^7$, и 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, поскольку эти шрифты и так мелкие).

Если Вы хотите изменить стиль набора формулы, можно в явном виде указать его с помощью 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»:

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

Наивная попытка набрать эту формулу выглядит так:

$\frac{7}{25} = \frac{1}{3 + \frac{1}{1 + \frac{1}{1 + \frac{1}{3}}}}$	<pre> $\frac{7}{25} = \frac{1}{3 + \frac{1}{1 + \frac{1}{1 + \frac{1}{3}}}}$ </pre>
--	--

Результат несколько несуразен, но формально полностью согласуется с правилами \TeX : 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. Правильно действовать следующим образом:

```


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


```

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

4.3. Фантомы и прочее

В разделе про набор матриц мы столкнулись с командой `\lefteqn`, позволяющей напечатать фрагмент формулы и при этом сообщить Т_ЭХу, 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_ЭХ надо обмануть: добавить в подкоренные выражения по символу, который чуть выше, чем a или d , чтобы подкоренные выражения оказались одной высоты. Этот символ, естественно, не должен печататься и не должен занимать места по горизонтали (лишние пробелы под корнем нам тоже ни к чему). Такой невидимый символ генерируется 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, занимающий по высоте столько же места, сколько открывающая скобка, и не имеющий ширины.

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

``

то результат будет такой же, как если бы эта самая «какая-то формула» была сначала напечатана по всем правилам Т_ЭХа, а затем аккуратно стерта с бумаги (но следов ластика Вы не увидите). Пример:


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

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

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 вписать формулу вручную.	можно	На пустое место <code>\hphantom{\sin^2\alpha}</code> 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_ЕXу, 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:

Вася + Маша = 	\$\$ <code>\mbox{Вася}+\mbox{Маша}</code> <code>=\lefteqn{\,\heartsuit}</code> <code>{\swarrow}</code> \$\$
---	---

Тонкий пробел перед `\heartsuit` мы поставили, чтобы сердце было более удачно расположено относительно стрелы; команду `\swarrow` мы взяли в фигурные скобки, чтобы Т_ЕX рассматривал ее как обычный символ (см. раздел II.4.4 ниже; в противном случае перед стрелкой будет автоматически сделан дополнительный пробел, также нарушающий взаимодействие сердца и стрелы).

4.4. Снова об интервалах в формулах

Сейчас мы обсудим вкратце, какими правилами руководствуется Т_ЕX при расстановке интервалов в математических формулах. Когда мы пользуемся при наборе формул готовыми Л_АT_ЕXовскими командами, об этом можно не думать, поскольку интервалы, скорей всего, будут и так правильными. 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 стр. 51) и при этом не хотим подбирать верные интервалы экспериментально.

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

Обыкновенный символ	(например, α);
Бинарная операция	(см. стр. 31);
Бинарное отношение	(см. стр. 31);
Математический оператор	(см. стр. 32, 34);
Подформула;	
Знак препинания;	
Скобка.	

Здесь подформула — это любой фрагмент формулы, заключенный в фигурные скобки, знак препинания — это запятая или точка с запятой.¹ К бинарным отношениям (с точки зрения

¹А также двоеточие, если оно задано командой `\colon`, и точка, если она задана командой `\ldotp`.

TeX) относятся также все стрелки (стр. 32), а также фрагменты формул, создаваемые командой `\stackrel`. При расстановке пробелов в формуле TeX руководствуется тем, к какому из типов 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. Подформула (то есть фрагмент формулы, заключенный в фигурные скобки) рассматривается TeXом 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 в главе I, когда обсуждали, как задать на TeXe десятичную дробь. Мы не будем вдаваться в точные правила расстановки пробелов (они перечислены в книге [2]). Для нас сейчас важнее то, что TeX можно заставить рассматривать любой фрагмент формулы как бинарную операцию, бинарное отношение или математическую операцию: для этого надо применить команды `\mathbin`, `\mathrel` или `\mathop` соответственно. Вот примеры того, как работают эти команды.

Иногда возникает нужда в символе $\hat{\otimes}$, рассматриваемом как символ бинарной операции. Естественно, этот символ можно сгенерировать, написав `\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 $E\otimes F$.

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

Чтобы TeX 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 $E\otimes F$.

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 главы VII по поводу того, как это сделать).

Типичный пример использования команды `\mathop` — определение имени операции, записываемой прямым шрифтом (см. стр. 32, где мы давали определение функции `tg`). Обозначения такого типа встречаются в математических текстах очень часто, и набора команд для них, предусмотренного L^ATeXом (стр. 32), вполне может не хватить; в этом случае, чтобы получить на печати, скажем, $\text{Ext}^1(E, F)$, надо написать:

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

Здесь `\mathop` необходимо для того, чтобы между именем «оператора» и тем, к чему он прилагается, автоматически вставлялся маленький дополнительный пробел, делающий формулу более читаемой:

Сравните `\sin x` и `{\rm sin}x`.
`\sin x` и `{\rm sin}x`.

Что же касается `\nolimits`, то эта команда необходима для того, чтобы в выключных формулах (точнее, в «выключном стиле» — см. раздел 4.2) верхние и нижние индексы к «оператору» записывались именно как индексы, а не над и под ним, как «пределы» (см. стр. 34).

А вот пример, когда Т_ЕХу надо объяснить, что некоторый сложный символ есть символ математического оператора. Предположим, нам понадобилась формула наподобие

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

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

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

В самом деле, из сказанного на стр. 35 вытекает, что наша запись равносильна такой:

$$\text{\$ \$} \quad \text{\sum}^{\prime}_{x \in \Gamma} f(x) \quad \text{\$ \$}$$

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

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

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_ЕХом 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.

Приведем, наконец, пример, в котором приходится использовать команду `\mathrel` (а также `\mathop`, причем не по прямому назначению). Иногда для того, чтобы указать, что одно множество «строго содержится» в другом, используют значок \subsetneq . Как его создать? Часть ответа должна быть уже ясна: поскольку это — знак бинарного отношения, что для того, чтобы вокруг него оставались правильные пробелы, надо написать

`\mathrel{\dots}`

А что именно написать на месте многоточия? Если бы знак неравенства должен был быть *над* знаком включения, сгодились бы команда `\stackrel{>}{\subset}`; а так мы воспользуемся таким трюком: превратим знак \subset из бинарного отношения в математический оператор с пределами и зададим знак \neq как его нижний предел:

```
\mathop{\subset}\limits_{\neq}
```

Окончательно получаем: чтобы создать формулу $E \subsetneq F$, надо написать

```
\mathrel{\mathop{\subset}\limits_{\neq}}F
```

Команда `\limits` необходима, иначе в текстовых формулах получилось бы не \subset , а \subsetneq (вспомните, что «пределы» математических операторов в текстовом стиле ставятся, при отсутствии команды `\limits`, не над и под оператором, а там же, где индексы).

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

4.5. Дополнительные пробелы вокруг формул

Пришла пора и для последней из тех тонкостей набора формул, о которых упоминается в нашей книге. Некоторые авторы и издатели считают, что математический текст выглядит понятнее, когда каждая формула окружена дополнительными пробелами справа и слева от нее. Для этих целей в \TeX е предусмотрен параметр `\mathsurround` (см. раздел I.2.6 по поводу \TeX овских параметров). Значение этого параметра — размер дополнительного пробела, вставляемого по обе стороны каждой внутритекстовой математической формулы (этот пробел не добавляется перед формулой, попавшей при печати в начало строки, и после формулы, попавшей в конец строки). При запуске \LaTeX а значение этого параметра равно нулю, так что расстояния между формулами и окружающим текстом такие же, как между словами в тексте. Можно, однако, присвоить параметру `\mathsurround` значение ненулевой длины. Например, если сказать в преамбуле

```
\mathsurround=2pt
```

то каждая формула будет окружена дополнительными пробелами по 2 пункта с обеих сторон.

Глава III.

Набор текста

1. Специальные знаки

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

1.1. Дефисы, минусы и тире

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

Чтобы получить на печати дефис, короткое тире или длинное тире, надо в исходном тексте набрать один, два или три знака – соответственно. В русских текстах рекомендуется использовать длинное тире в качестве тире как такового, а короткое тире — в сочетаниях типа «я вернусь через 2–3 часа» (в исходном тексте это выглядит как **через 2--3 часа**; обратите внимание на отсутствие пробелов вокруг тире). Длинное тире, напротив, должно быть окружено пробелами с обеих сторон (этого требует не Т_ЕХ, но принятые в России типографские правила).

Знак минуса, в отличие от короткого тире, встречается только в математических формулах, и там он, как Вы помните, изображается просто знаком – (см. раздел I.3).

У любознательного читателя может возникнуть вопрос, как так получается, что запись **жж** в исходном тексте дает на печати всего-навсего две буквы «жж», а запись **--** дает тире, которое шире, чем два дефиса. Ответ: Т_ЕХовские шрифты так устроены, что некоторые последовательности подряд идущих символов заменяются на печати на новый знак (говоря более техническим языком, в этих шрифтах допускаются лигатуры).

Наряду с тире, другой пример лигатур — это то, как выглядит в основных шрифтах сочетание букв fi: не так, как поставленные рядом f и i.

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).

2.2. Промежуток промежутку рознь

В обычном режиме \TeX выравнивает справа строки абзаца, при необходимости делая переносы и слегка растягивая или сжимая промежутки между словами. Промежутки между предложениями при этом сами по себе шире и являются более растяжимыми, чем между словами внутри предложения. Посмотрите внимательно на следующий пример:

Ура! Мы победили. Посмотрим, что будет в следующем матче.

Чтобы отличить промежутки между словами от промежутков между предложениями, \TeX не проводит синтаксического разбора, но применяет следующие правила:

1) Пробел увеличивается после:

- точки, вопросительного знака, восклицательного знака (в максимальной степени);
- двоеточия (несколько меньше);
- точки с запятой (еще меньше);
- запятой (совсем чуть-чуть).

2) Если последняя из букв, встретившихся перед одним из упомянутых в пункте 1 знаков препинания, была прописной, то пробел после этого знака препинания не увеличивается.

3) Если после одного из упомянутых в пункте 1 знаков препинания следует закрывающая скобка (круглая или квадратная) или закрывающие кавычки (любые), а затем — пробел, то этот пробел увеличивается.

Смысл правила 2 в том, что точка после прописной буквы чаще всего обозначает не конец предложения, а конец чьих-то инициалов.

Как это и бывает обычно с «машинными эвристиками», сформулированные правила иногда приводят к неверным результатам: точка после строчной буквы может встретиться и в середине предложения, например, в сокращении, а точка после прописной буквы может, напротив, попасть в конец предложения. В этих случаях надо следующим образом помочь \TeX у сделать правильные пробелы:

- Если точка после строчной буквы *не* заканчивает предложения, то после нее следует поставить команду `\` (backslash с пробелом), генерирующую обычный пробел между словами (см. стр. 12).
- Если точка (или любой другой из перечисленных в пункте 1 знаков препинания) после прописной буквы заканчивает предложение, то перед ней следует поставить команду `\@` — тогда пробел после точки будет обычным образом увеличен.

Вот пример:

Многие (в том числе гр. Иванов И.И.) думают, что \TeX овские правила для пробелов непросты. Иное мнение имеет Петров П.П. Ему все это легко.

Многие (в том числе гр. \ Иванов~И.И.)\ думают, что \TeX{}овские правила для пробелов непросты. Иное мнение имеет Петров~П.П\@. Ему все это легко.

Обратите внимание, что в этом примере мы поставили backslash с пробелом и после закрывающей скобки. Это пришлось сделать, так как закрывающая скобка хоть сама пробелы и не увеличивает (правило 1), но и точке это делать не мешает (правило 3).

Наконец, последнее правило относительно увеличения пробелов: если пробел задан как неразрывный с помощью символа ~, то он не увеличивается, невзирая ни на какие предшествующие знаки препинания. Кстати, и в вышеприведенном примере лучше было бы написать гр. ~Иванов.

Если Вам хочется, чтобы все пробелы между словами были одинаковы, то можно с помощью команды

```
\frenchspacing
```

вообще отменить особый статус промежутков после концов предложений. После этого команды \@ и \ для настройки промежутков между словами станут ненужными (если Вы их все-таки употребите, это ни на что не повлияет), и Вы сможете благополучно забыть про все тонкости из этого раздела. Другой вопрос, в каком случае текст выглядит красивее. Строгих правил тут нет, в значительной мере это дело вкуса.

Восстановить режим, при котором промежутки между предложениями больше промежутков между словами, можно с помощью команды

```
\nonfrenchspacing
```

Обе эти команды можно употреблять в любом месте исходного файла, но разумное место для них — в преамбуле.

2.3. Установка промежутков вручную

Как команда «backslash с пробелом» \ , так и символ неразрывного пробела ~ генерируют пробел, но делать пробелы вручную с помощью набора чего-нибудь вроде ~~~ или \ \ \ неразумно, поскольку эти пробелы, как правило, могут растягиваться или сжиматься ради выравнивания строк, и Вы не сможете проконтролировать реальный размер пустого пространства, полученного таким способом.

Чаще всего требуется получить промежуток величиной в один или два em (см. стр. 16). Для этого служат команды \quad, дающая промежуток в 1 em, и \qqquad, дающая промежуток в 2 em. 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 разделе 1.2.

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. стр. 16), например, \hspace{1.5cm}

Пользуясь командами типа \quad, не сделайте лишнего пробела (разумеется, кроме того, который ограничивает имя команды, состоящей из букв). Вот примеры того, как надо и как не надо делать:

Здесь 1em промежутка.
 Здесь 1em промежутка.
 Здесь больше, чем надо.

Здесь\quad 1em промежутка.\
 Здесь\quad{}1em промежутка.\
 Здесь\quad{} больше,
 чем надо.\
 \

В этом примере используется еще команда `\`, начинающая новую строку (см. подробности на стр. 73).

3. Диакритические знаки и прочее

Во многих языках используются буквы с дополнительными значками, размещающимися над или под буквой (они называются диакритическими знаками). Кроме того, в ряде языков, использующих латинский алфавит, есть специальные дополнительные буквы. В `TeX` имеются команды для набора таких букв с диакритическими знаками из почти всех европейских языков. Команды для получения диакритических знаков собраны в нижеследующей таблице, где знаки проставлены, для примера, при букве «e».

Набрано	Вышло	Набрано	Вышло
<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>	ë	<code>\v{e}</code>	ě
<code>\H{e}</code>	ĕ	<code>\"e \"</code>	ë
<code>\c{e}</code>	ç	<code>\d{e}</code>	ę
<code>\b{e}</code>	ē	<code>\t oo</code>	ō

В следующей таблице Вы найдете команды для набора букв специального вида, а также перевернутых вопросительного и восклицательного знаков (последние используются в испанском языке).

Набрано	Вышло	Набрано	Вышло
<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>	Ƶ
<code>\ss</code>	ß		
<code>!'</code>	¡	<code>?'</code>	¿

Обратите внимание на команды `\i` и `\j` в этой таблице: они нужны для того, чтобы ставить диакритические знаки над буквами *i* и *j*. Если просто сказать, допустим, `\=i`, то получится \bar{i} , а это не то, что требуется. Правильно писать `\=i`. Вот несколько примеров.

Oui, c'est peut-être ça. Ёжик	Oui, c'est
под ёлкой.	peut- <code>\^</code> etre <code>\c{c}</code> a.
¡Богатые тоже плачут!	<code>\"Ежик под \"елкой.</code>
	!‘Богатые тоже плачут!

Для набора русских текстов из всего этого великолепия нужны фактически только две команды: `\’` (чтобы ставить ударения) и `\"` (для буквы ё).

Команды `\’`, `\c` и т. д. имеют один обязательный аргумент — букву, над (или под) которой надо ставить диакритический знак. Внимательный читатель может заметить тут противоречие со сказанным на стр. 15: ведь обязательный аргумент должен быть заключен в фигурные скобки, а в нашей таблице в записях вроде `\’e` или `\"e` фигурных скобок нет. На самом деле противоречия нет, просто на стр. 15 было сказано не все: если у Л^AT_EXовской команды предусмотрен один обязательный аргумент и в исходном тексте после имени команды непосредственно следует буква, то в качестве аргумента будет воспринята именно эта буква. Так что можно было бы, в принципе, не ставить букву в фигурные скобки и при командах наподобие `\c` или `\u` (но психологически приятнее, когда слово, которое на печати выйдет без пробелов, не будет содержать пробелов и в исходном тексте):

façade или façade — все равно.	<code>fa\c{c}ade</code> или
	<code>fa\c cade</code> ---
	все равно.

Команды из второй таблицы аргументов не требуют; что же касается значков !‘ и ?‘, то это вообще не команды, а лигатуры (см. текст мелким шрифтом на стр. 58).

В стандартном наборе есть команда `\aa`, дающая на печати å , но нет команды для постановки аналогичного диакритического знака над любой буквой. Средствами T_EXа такую команду нетрудно создать. Например, она определена в русифицирующем стиле `ruscorr.sty`, описанном в приложении Б.

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 разделе II.2.7.

В заключение нашего обсуждения диакритических знаков отметим вот что. Хотя описанные в этом разделе команды дают возможность набирать все знаки алфавита практически любого европейского языка, это еще не значит, что Вы так просто сможете набирать на T_EXе длинные тексты на этих языках. Дело в том, что при отсутствии ориентированной на соответствующий язык таблицы переносов T_EX будет, скорее всего, переносить слова в этих текстах неправильно. Таблица переносов для русского языка в русификации Л^AT_EXа имеется, а для английского языка такая таблица присутствует в T_EXе изначально. Если же Вам придется набирать хотя бы абзац текста на каком-то другом языке, лучше всего от греха подальше переключиться на режим, в котором переносов делаться вообще не будет. Как это сделать, будет рассказано в разделе, посвященном абзацам.

4. Переключение шрифтов

В первой главе мы уже сталкивались с командами, изменяющими текущий шрифт. Давайте рассмотрим их систематически.

Начнем мы с такого предупреждения, которое не требуется профессиональным полиграфистам, но не повредит остальным читателям:

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

Обычно шрифт, отличный от используемого в основной части текста, применяется для выделения каких-то частей этого текста. Например, шрифтом выделяют заголовки разделов; по этому поводу Вам беспокоиться незачем, поскольку для таких выделений \LaTeX выбирает шрифт автоматически (если, разумеется, Вы оформляете разделы текста с помощью команд, описываемых в следующей главе, а не пытаетесь сделать это вручную). Кроме того, может потребоваться выделить шрифтом не заголовок, который \LaTeX делает сам, а какую-то выделенную Вами часть текста. На этот случай в \LaTeX предусмотрена команда `\em`. Если текущий шрифт прямой, то эта команда переключает шрифт на курсивный, а если имеет наклон, то на обычную гарнитуру `roman`. Поскольку в некоторых ситуациях \LaTeX выбирает шрифт за Вас (в заголовках, колонтитулах, в текстах «теорем» и т. п.), рекомендуется именно эту команду использовать в первую очередь для выделения текста.

Употребляя команду `\em` (а также команды, устанавливающие «наклонный» шрифт), необходимо учитывать, что при соседстве слова, набранного шрифтом с наклоном (курсивом в частности) и слова, набранного прямым шрифтом, последняя буква «наклонного» и первая буква «прямого» слова могут слишком сблизиться, что на печати выглядит некрасиво. Взгляните еще раз на стр. 14, где последняя буква слова «восстановится» сближается с первой буквой слова «шрифт». 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. Если последним печатным знаком в шрифте, имеющем наклон, был знак препинания, то после него ставить `\/` не нужно: требуемый эффект достигается за счет места, занимаемого в тексте этим знаком. Проиллюстрируем все сказанное примером.

Этот текст *выделен; внутри выделенного текста можно тоже делать* выделения *тем же способом; теперь снова* обычный текст. Если выделенный текст *кончается запятой*, то предосторожности не нужны.

Этот текст `{\em выделен;`
внутри выделенного текста
можно тоже делать
`{\em выделения\/}` тем же способом;
теперь снова`\/}` обычный текст.
Если выделенный текст *кончается*
`{\em запятой,}` то предосторожности
не нужны.

Если команда `\/` окажется после текста, который в результате действия `\em` оказался набран прямым шрифтом, то никакого вредного действия это не окажет.

Если команда `\/` поставлена между двумя буквами, дающими на печати лигатуру (см. стр. 58), то вместо лигатуры на печати получатся две эти буквы по отдельности; если эту команду поставить в слове между двумя буквами, между которыми в текущем шрифте предусмотрен кернинг, то кернинг между этими буквами будет отключен.

Теперь перечислим все существующие в стандартном комплекте \LaTeX гарнитуры (виды шрифта) и кегли (размеры). Начнем с гарнитур. В нижеследующей таблице они перечислены вместе с командами, задающими переключение на них.

Команда	Название гарнитуры
<code>\bf</code>	полужирный шрифт (boldface)
<code>\it</code>	<i>курсив (italic)</i>
<code>\sl</code>	<i>наклонный шрифт (slanted)</i>
<code>\sf</code>	рубленный шрифт (sans serif)
<code>\sc</code>	КАПИТЕЛЬ (SMALL CAPS)
<code>\tt</code>	имитация пишущей машинки (typewriter)
<code>\rm</code>	«обычный» шрифт (roman)

Шрифты гарнитуры `\tt` обладают специфическими свойствами: все символы в них имеют одинаковую ширину, промежутки между словами жестко фиксированы и не обладают растяжимостью, в словах, набранных этими шрифтами, не делается автоматических переносов при верстке абзацев. Применяются эти шрифты для изображения компьютерных программ, фрагментов Т_EXовских исходных текстов и т. п.

А теперь перечислим возможные размеры шрифтов:

Команда	Название размера
<code>\tiny</code>	Маленький-маленький
<code>\scriptsize</code>	Очень маленький (как индексы)
<code>\footnotesize</code>	Маленький (как сноски)
<code>\small</code>	Маленький
<code>\normalsize</code>	Нормальный
<code>\large</code>	Большой
<code>\Large</code>	Еще больший
<code>\LARGE</code>	Очень большой
<code>\huge</code>	Совсем большой
<code>\Huge</code>	Ужасно большой

Примеры использования команд переключения шрифтов Вы видели на стр. 14.

Команды, меняющие размер, одновременно устанавливают гарнитуру roman («обычный» шрифт). Поэтому полужирный большой шрифт требует не команд `\bf\large`, а команд `\large\bf`.

Новейшие версии Л^AT_EXа снабжены так называемой «новой схемой выбора шрифта» (NFSS), в которой перечисленные команды для выбора гарнитуры и кегля имеют несколько иной смысл. См. приложение В.

Скорее всего, Вы будете давать команды, переключающие шрифт, внутри группы. При этом не сделайте лишнего пробела до или после фигурной скобки, как в следующем примере:

Вот **так** получится плохо.
Вот **так** будет правильно.

Вот `{\bf так }` получится плохо. \\
Вот `{\bf так}` будет правильно.

Реальный размер шрифтов, задаваемых командами `\large`, `\small` и т. п. зависит от стиля. В стандартных стилях с основным шрифтом кегля 12 команды `\huge` и `\Huge` задают один и тот же размер (кегель 25).

С командами, меняющими размер шрифта, связана одна распространенная ошибка. Если Вы набрали шрифтом измененного размера (скажем, `\small`) целый абзац, то в момент, когда `TeX` видит пустую строку (или команду `\par` — см. стр. 78), этот шрифт *не* должен быть еще переключен на обычный, иначе интервалы между строками получатся неправильные. Вот пример того, как надо и как не надо действовать:

Мы закрываем группу и возвращаемся к обычному шрифту только после пустой строки, завершающей абзац.

Здесь мы вернулись к обычному шрифту раньше времени. Результат — межстрочные интервалы слишком велики.

```
{\footnotesize
Мы закрываем группу и
возвращаемся к обычному
шрифту только после пустой
строки, завершающей абзац.
```

```
}
{\footnotesize
Здесь мы вернулись к обычному
шрифту раньше времени. Результат
--- межстрочные интервалы
слишком велики.}
```

Для любознательных объясним, откуда берется этот эффект. Команды, меняющие размер шрифта, изменяют не только текущий шрифт, но и расстояние между строками (увеличивают, если размер шрифта увеличивается, и уменьшают, если размер шрифта уменьшается). Однако же определение расстояний между строками происходит только на заключительном этапе верстки абзаца. 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. Сноски

Чтобы сделать сноску к какому-то месту в тексте, достаточно использовать команду `\footnote` с одним обязательным аргументом — текстом сноски. В стандартных стилях `LaTeX` сноски¹ нумеруются подряд на протяжении всей главы или даже (в стиле `article`) всего документа. В исходном тексте предыдущий фрагмент выглядел так:

```
сноски\footnote{Вроде этой} нумеруются ...
```

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

Если после слова, к которому делается сноска, должен стоять знак препинания, то в исходном тексте его надо поставить *после* закрывающей фигурной скобки, ограничивающей аргумент команды `\footnote`.

Текст сноски может состоять из нескольких абзацев; в этом случае они, как обычно, разделяются пустой строкой.

К сожалению, не существует разумного способа автоматически нумеровать сноски так, чтобы нумерация начиналась заново на каждой странице. Если Вы готовы пожертвовать возможностью автоматической нумерации сносок, то можно воспользоваться командой `\footnote`

¹Вроде этой

с необязательным аргументом. Этот необязательный аргумент ставится (в квадратных скобках) перед обязательным³³. Предыдущий фрагмент выглядел в исходном тексте так:

```
обязательным\footnote[33]{Видите, какой ... }
```

При использовании команды `\footnote` с необязательным аргументом автоматическая нумерация сносок «не сбивается»: предыдущая сноска имела номер 1, затем мы искусственно создали сноску номер 33, а следующая сноска² будет иметь номер 2.

В случае, если Вы хотите сделать сноску к тексту, входящему в «блок» (например, в аргумент команды `\mbox`; в главе VIII мы расскажем о том, что такое блок в общем случае и какие команды генерируют блоки), команда `\footnote` непригодна. Вот как надо ставить сноски в этом случае:

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

В этом случае получится нормальная сноска, информирующая нас, что Роман «Три³ мушкетера» написал Дюма. Если бы мы просто написали `\footnote`, то увидели бы на печати только номер, но не саму сноску.

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`, а второй раз — в качестве необязательного аргумента команды `\footnotetext` (необязательный аргумент этой команды должен идти *перед* обязательным):

```
Роман \mbox{\лк Три\footnotemark[99]}
мушкетера}\footnotetext[99]{А не четыре!}
написал Дюма.
```

6. Абзацы

Чтобы Т_ЕX сверстал абзац, никаких специальных усилий прилагать не нужно: достаточно оставить в исходном тексте пустую строку, указывающую Т_ЕXу на конец абзаца. В этом разделе речь пойдет о тех «нештатных ситуациях», которые иногда при этом возникают.

Обычно абзацы делаются выровненными по правому краю; при необходимости промежутки между словами растягиваются или сжимаются, а в словах делаются переносы. Т_ЕX выбирает из всех вариантов разбиения текста абзаца на строки оптимальный; при этом и для сжатия, и для растяжения промежутков между словами есть пределы, которые Т_ЕX старается не превышать.

Если это удастся, то получается аккуратный абзац. Нас же сейчас больше волнует, что происходит в том случае, когда это *не* удастся.

Прежде всего заметим, что «предел сжимаемости» строки не превышает Т_ЕXом ни при каких условиях⁴; что бы ни было, строки не станут более тесными, чем им позволяют параметры шрифта. Поэтому строки, которые не удалось ужать, остаются чересчур длинными

³³Видите, какой интересный номер получился!

²Вот эта.

³А не четыре!

⁴Если не предпринимать для этого специальных усилий.

(при этом, естественно, они выходят на поля документа). С другой стороны, предел растяжимости, за неимением ничего лучшего, может быть превышен. При этом получается то, что полиграфисты называют жидкими, или разреженными строками:

Весьма и весьма жидкая строка.

О каждой из этих двух неприятностей Т_ЭX сообщает в процессе трансляции (эти сообщения появляются на экране, и, кроме того, записываются в log-файл — специальный файл с тем же именем, что у файла, который обрабатывался системой, и расширением log).

Предположим, Вы получили строку, выходящую на поля, например, такую (в нашем примере она отмечена черным прямоугольником):

Еще одно правило относительно увеличения пробелов: если пробел задан как неразрывный, то он не увеличивается, невзирая ни на какие предшествующие знаки препинания. Некоторые полиграфисты считают, что в русских текстах пробелы между словами и между предложениями должны всегда быть одинаковыми.

При получении строк, выходящих на поля, выдается на экран и записывается в файл с расширением log сообщение наподобие такого:

```
Overfull \hbox (1.77339pt too wide)
in paragraph at lines 7--12
[]\elvrm ... уве-ли-че-ния про-бе-лов: если

\hbox(7.54332+2.12917)x310.72488, glue set - 1.0
.\hbox(0.0+0.0)x17.0
.\elvrm Е
.\elvrm ш
.\elvrm е
.\glue 3.65 plus 1.825 minus 1.21666
.etc.
```

Для нас с Вами существенна информация, содержащаяся в первых трех строках этого сообщения (дальнейший загадочный текст, выдающийся только в log-файл, но не на экран, описывает, как именно «сделана» неудачная строка; такие вещи могут представлять интерес только для профессиональных Т_ЭXников). Давайте разберем, что в них написано. Сначала идет надпись «Overfull \hbox», указывающая, что произошло «переполнение» (overfull) строки. В скобках указано, на какое именно расстояние строка выходит за край: на 1,77339 пунктов. (Напомним, что пункт примерно равен одной трети миллиметра — см. стр. 16.) Далее сказано, что overfull произошел при верстке абзаца (слова «in paragraph»), а затем указаны номера строк исходного файла, в которых был записан этот абзац.

Наконец, в третьей строке в этом сообщении приведен фрагмент неудачной строки вблизи ее конца (конца не в исходном тексте, а на печати). Обратите внимание, что в некоторые слова вставлены дефисы: они показывают те места, в которых Т_ЭX в принципе мог бы сделать переносы. Когда мы взглянем внимательнее, то станет ясно и причина катастрофы: в слове **если**, которым заканчивается строка, дефиса не стоит вообще; значит, Т_ЭX не смог найти в нем подходящего места для переноса и оказался 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 жидких строках выглядят так:


```
Underfull \hbox (badness 2318)
in paragraph at lines 940--942
\elvrn мен-ты спис-ка ну-ме-ру-ют-ся
```

(в log-файле эти сообщения имеют и продолжение, аналогичное тому, что мы видели в сообщении об overfull'e; мы не стали воспроизводить его еще раз). Главных элементов в этих сообщениях три:

- Само слово Underfull, указывающее, что речь идет о жидкой строке;
- Указание на строки исходного текста, в которых находится абзац с жидкой строкой (в нашем случае 940–942);
- Численная характеристика того, насколько разрежена строка (по-английски это число называется badness). В нашем случае это число равно 2318; вскоре мы обсудим, что оно значит.

Итак, мы выяснили, какие могут быть неприятности при верстке абзацев и как Т_ЕX сообщает нам об этих неприятностях. Вся оставшаяся часть этого раздела посвящена тому, как с этими неприятностями бороться.

6.1. Переносы

Вероятно, читатель был шокирован тем, что в нашем примере с overfull'ом Т_ЕX не смог найти, где сделать перенос в слове «если». Дело, в частности, в том, что обычно Т_ЕX не делает переносов, при которых от слова отрываются две последние буквы. Это соответствует нормам английской орфографии, но для русского языка такое ограничение неуместно. Поэтому при работе с русскими текстами можно (и нужно) установить такой режим, в котором перенос двух последних букв допустим. Устанавливается этот режим с помощью изменения параметра `\righthyphenmin` (см. стр. 14 по поводу Т_ЕXовских параметров). Значение этого параметра — целое число, равное наименьшему количеству букв в слове, которые можно переносить на следующую строку. Стало быть, если написать в файле

```
\righthyphenmin=2
```

то при обработке всех абзацев, кончающихся после этой команды, переносы с отрывом двух последних букв будут разрешены. Если Вам встретится абзац английского текста, то перед завершающей его пустой строкой дайте команду

```
\righthyphenmin=3
```

дабы не сделать неверных переносов в английских словах.

Общее облегчение режима переносов с помощью `\righthyphenmin` может, тем не менее, ничем не помочь против встретившегося Вам overfull'a: не исключено, что Т_ЕX действительно не знает, как перенести какое-то слово. Таких слов не очень много, но они встречаются. Кроме того, Т_ЕX *не* делает автоматических переносов в словах, содержащих диакритические знаки (для русского языка — ударения над буквами или букву ё), а также в словах, в которых присутствуют наряду с буквами цифры, знаки препинания и т. п. Далее, если в слове присутствует дефис, то Т_ЕX сможет сделать в нем перенос только на месте этого дефиса (слово «генерал-губернатор» будет перенесено так, что генерал- останется в конце строки,

а губернатор начнет следующую строку). Что делать в таких случаях, когда автоматически вставляемых Т_ЕХом переносов не хватает?

Совет номер один — попробуйте немного переписать абзац, изложив в нем мысли по-другому. Обычно после небольших изменений в абзаце неприятности с переносами исчезают; как показывает опыт, качество текста при этом тоже зачастую улучшается (с точки зрения языка, а не Т_ЕХа). Пусть, однако, улучшать изложение дальше некуда, а абзац все равно получается неудачный. Посмотрим, что еще можно сделать, чтобы избавиться от `overflow`'а.

Если Т_ЕХ не может перенести слово, перенос которого по правилам русского языка возможен, то есть два способа указать Т_ЕХу на это обстоятельство.

Во-первых, существует «одноразовый» способ указать Т_ЕХу места переносов в слове. Это делается с помощью команды `\-` таким, например, образом:

```
на\-\ "ем\-\-ник
```

Команда `\-` означает, что данное слово можно переносить в тех и только тех местах, где стоят знаки `\-` (хотя бы и вопреки тому, что диктует Т_ЕХовский алгоритм переноса). Она годится для любых слов (с буквой ё, цифрами и т. д.) Однако при этом Т_ЕХ не запоминает, какие слова и в каких местах позволила ему перенести команда `\-`. Если, например, то же слово «наёмник» встретится в тексте еще раз, Т_ЕХ по-прежнему будет считать, что переносить его нельзя.

Во-вторых, если слово, которое надо переносить не в тех местах, которые находит алгоритм переноса, встречается в тексте неоднократно, имеет смысл указать это Т_ЕХу «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. Не забывайте, что конец строки — тоже пробел, так что слова можно располагать и в нескольких строчках. 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`.

Бывают случаи, когда избавиться от `overflow`'а не помогают даже идеально расставленные переносы: например, если в конец строки попадает слово, которое переносить нельзя, вроде «гвоздь», или не допускающая переноса математическая формула (кстати, если в неудачной строке присутствует нечто более сложное, чем просто текст, Т_ЕХ в сообщении об `overflow`'е заменяет это на пару квадратных скобок `[]`). Что делать в таких случаях, рассказано в следующих разделах.

6.2. Команда `\sloppy` и параметр `\emergencystretch`

Существует простой и грубый способ раз и навсегда избавиться от `overflow`'ов. Для этого достаточно включить в файл команду `\sloppy` — больше сообщений о слишком длинных

строках Вы, скорее всего, не увидите. Разумеется, просто так ничего не бывает. В режиме, включаемом этой командой, допустимы сколь угодно жидкие строки (Т_ЕX по-прежнему старается, чтобы их было поменьше и чтобы они были не очень жидкими, но если надо выбирать между `overflow`’ом и жидкой строкой, он обязательно выберет второе). Как будет выглядеть при этом абзац, заранее предсказать трудно. Какие-то абзацы могут смотреться приемлемо, какие-то — чудовищно. Поэтому ставить `\sloppy` в преамбулу, задавая тем самым этот режим на весь текст, несколько рискованно. Разумнее задать эту команду перед концом того абзаца, в котором произошел `overflow`, и посмотреть, что из этого получится.

Если Вы не хотите, чтобы действие `\sloppy` распространилось и дальше, 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 верстку абзаца. Правило таково:

Режим верстки абзаца определяется в тот момент, когда транслятор Т_ЕXа 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:

Черепеховый суп — изысканное деликатесное и диетическое блюдо	Черепеховый суп --- изысканное деликатесное и диетическое блюдо {\sloppy }
---	--

А вот — типичная ошибка начинающего:

Черепеховый суп — изысканное деликатесное и диетическое блюдо	{\sloppy Черепеховый суп --- изысканное деликатесное и диетическое блюдо}
---	--

Группа завершилась до пустой строки, поэтому к моменту, когда Т_ЕX, увидев эту пустую строку, получил команду «сверстать абзац», он снова был в стандартном режиме, и вместо желаемых жидких, но не выходящих за край строк случился `overflow` (в первой строке).

Существует также способ не концентрировать всю разреженность в одной строке, как может получиться в результате действия команды `\sloppy`, а распределить ее более или менее равномерно по всему абзацу. Для этих целей используется параметр `\emergencystretch`. Это — некоторая длина, по умолчанию равная нулю. Ниже мы объясним точный смысл этого параметра, но для первого раза достаточно запомнить, что, если установить его значение равным примерно 3–5 пунктам, то есть написать, например,

```
\emergencystretch=5pt
```

то в случае, когда без `overflow`’ов сверстать абзац не удастся, Т_ЕX попытается сделать *все* строки абзаца более разреженными (тем более разреженными, чем больше величина этого параметра). Точную величину `\emergencystretch` надо подбирать экспериментально.

6.3. Ручное управление разрывами строк

Иногда возникает необходимость повлиять на то, в каких местах Т_EX начинает новую строку при верстке абзаца. Для этой цели есть соответствующие команды, и с одной из них мы уже встречались: это «неразрывный пробел», позволяющий запретить разрыв строки между двумя словами.

Бывает, надо обеспечить, чтобы в каком-то слове не делалось переносов, причем не вообще никогда (тогда разумно применить команду `\hyphenation`), а только в данном месте. Для таких целей удобно применить команду `\mbox` следующим, например, образом:

Параметр <code>filename</code> задает имя файла.	Параметр <code>\mbox{\bf filename}</code> задает имя файла.
--	---

Команда `\mbox` имеет один обязательный аргумент: в фигурных скобках может находиться любой текст (в том числе, как Вы заметили, с командами переключения шрифта и т. п.); Т_EX 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».

Есть еще один, несколько хулиганский, способ запретить Т_EXу делать перенос в данном слове и в данном месте: надо в конце слова без пробела поставить команду `\-`, например, так:

Слово `корова\-` в данном своем вхождении перенесено не будет.

Автор позаимствовал это прием из книги [5].

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 возможный вариант — 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 разрезанная.	Эта строка <code>\</code> была разорвана. Справа осталось пустое место, но зато строка не разрезанная.
---	--

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 растянутой, то Вы к тому же получите и сообщение об этом во время трансляции.

Команда `\` допускает и необязательный аргумент (см. стр. 15): если в квадратных скобках указать какое-то расстояние (в Т_EXовских единицах длины — стр. 16), то после оборванной строки будет оставлено это расстояние (по вертикали). Пример:

Разорвем строку

и оставим место.

При использовании команды `\` с необязательным аргументом бывает удобно вместо расстояния в явном виде указать один из следующих параметров:

`\smallskipamount` Маленький вертикальный пробел;

`\medskipamount` Вертикальный пробел побольше

`\bigskipamount` Еще больше.

Точный размер этого пробела зависит от стиля документа; на стр. 79 изображена величина соответствующих пробелов в стандартных стилях со шрифтом кегля 12.

Команда `\` имеет и вариант со звездочкой (см. раздел I.2.9); 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 может 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 Т_ЭХовский алгоритм верстки абзаца, делая для него разрыв в указанном месте все более выгодным, невзирая на возможное появление жидких строк.

Есть также команда `\nolinebreak`, действующая противоположно; она также может принимать необязательный аргумент — целое число от 0 до 4. Будучи заданной без аргумента, эта команда запрещает разрыв строки в указанном месте; точно так же она действует, если ее необязательный аргумент равен 4. Когда ее необязательный аргумент возрастает от 1 до 3, возрастает невыгода разрыва строки в указанном месте, даже невзирая на то, что из-за отказа от этого разрыва могут появиться жидкие строки.

Для простых приложений, о которых идет речь в этой главе, команды `\linebreak` и `\nolinebreak`, как правило, не нужны: чтобы красиво разорвать строку, нужна команда `\`, а для запрещения разрыва гораздо удобнее «неразрывный пробел». По-настоящему эти две команды требуются только при разработке собственных макроопределений, о чем сейчас говорить преждевременно.

6.4. Верстка абзацев без выравнивания и переносов

Можно перевести Т_ЭХ в режим, при котором он вообще не будет пытаться выравнивать текст по правому краю и не будет 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 Т_ЭХом 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 `overflow`'ов.

Этот абзац мы сверстали
без выравнивания и переносов.
Может быть, вид и не очень
аккуратный, зато без
`overflow`'ов. `{\raggedright`

}

К сожалению, команда `\raggedright` в том виде, как она представлена в Л^AT_EXе, имеет не всегда желательное побочное действие: она делает абзацный отступ равным нулю. В вышеприведенном примере этого не произошло, поскольку команда `\raggedright` была выполнена после начала абзаца, когда абзацный отступ уже был определен; если, однако, записать ее в преамбулу, то отступ будет равен нулю для всех абзацев. Если Вам это не нравится, но выравнивать текст по правому краю все-таки не хочется, можно после `\raggedright` записать в преамбуле команду, устанавливающую значение абзацного отступа `\parindent` (см. стр. 14; в стандартных стилях значение этого параметра равно примерно 1,5em).

6.5. Более тонкая настройка

Режимы, задаваемые командами `\sloppy` и `\fussy`, представляют собой две крайности. Здесь мы расскажем Вам о более аккуратных способах управления версткой абзацев. При первом чтении этот раздел можно пропустить.

А. Параметр `\hfuzz`. Если Вы получаете слишком много сообщений об `overflow`'ах, можно попросить Т_EX вообще не считать как 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 `overflow` 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.

На первый взгляд такой способ борьбы с `overflow`'ами напоминает страусову политику: вместо того, чтобы преодолевать трудность, мы делаем вид, что ее не существует. Тем не менее, как показывает опыт, если `\hfuzz` равен примерно одному пункту, то получается вполне приемлемый результат. Дело, возможно, в том, что на фоне идеально выровненных абзацев одна выдающаяся на 1 пункт строка смотрится хуже, чем длинный текст, где все абзацы выровнены не идеально, а «с точностью до одного пункта».

Б. Мера разреженности строки. Как Вы помните, в сообщении Т_EXа 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 = 239</code>
Как может выглядеть разреженная строка.	<code>badness = 1558</code>
Как может выглядеть разреженная строка.	<code>badness = 2626</code>
Как может выглядеть разреженная строка.	<code>badness = 5022</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 Т_EXа, такие разреженные строки настолько плохи, что нет смысла делать различия между ними.

Для интересующихся объясним подробнее, как вычисляется `badness`. Как мы уже говорили в разделе 2.2, промежутки между словами в тексте не фиксированы, а могут растягиваться или сжиматься.

Каковы эти пробелы и насколько они могут растягиваться, зависит от шрифта (для примера: у основного шрифта кегля 10 обычный промежуток между словами равен примерно 3.33 пункта, и при этом он может растягиваться на 1.67 пунктов; промежуток же между предложениями в этом шрифте равен 13.33 пункта и может растягиваться на 5 пунктов)⁵. Когда \TeX растягивает строку с целью выравнивания, он находит сумму «пределов растяжимости» всех промежутков — это «предел растяжимости» строки, — и вычисляет, насколько требуемая длина строки больше «естественной» (определяемой размерами слов и нерастянутых промежутков между словами) — это «требуемое растяжение» строки. Отношение «требуемого растяжения» к «пределу растяжимости» строки определяет, насколько разреженной получится строка. Традиционно это отношение обозначается буквой r . Практически в качестве меры разреженности используется не само число r , а число $100r^3$ — это и есть *badness*. Если даже окажется, что $100r^3 > 10000$, *badness* все равно будет считаться равной 10000: строки, для которых отношение r больше или равно 4.7 (примерно при этом значении получается 10000), рассматриваются \TeX ом как одинаково плохие.

В том счастливом случае, когда требуемая длина строки совпадает с естественной, мера разреженности (*badness*) 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 \TeX ом как слегка разреженные. \TeX старается, чтобы такая «слегка разреженная» строка не попала в абзаце рядом со строкой, в которой промежутки между словами сжимались.

Теперь мы можем объяснить точный смысл параметра `\emergencystretch`. Если при верстке абзаца не удалось избежать *overfull*'а, то, при условии, что значение `\emergencystretch` отлично от нуля, \TeX делает еще одну попытку верстки, при которой в процессе перебора вариантов разбиения абзаца на строки (и вычислений соответствующих значений *badness*) к «пределу растяжимости» каждой из строк прибавляется значение `\emergencystretch`.

В. Параметр `\tolerance`. Теперь в нашем распоряжении есть все необходимые понятия, чтобы объяснить, как \TeX выбирает между разреженной строкой и *overfull*'ом (см. стр. 69)

При верстке абзаца \TeX *никогда* не создает строки, мера разреженности (*badness*) которых больше, чем значение параметра, называемого `\tolerance`. При невозможности удовлетворить этому условию создаются строки, выходящие за край: возникает *overfull*. С другой стороны, если мера разреженности строки не превосходит значения `\tolerance`, то будет создана именно столь разреженная строка, но не *overfull*.

В отличие от некоторых других систем компьютерной верстки, \TeX *никогда* не растягивает и не сжимает отдельное слово.

В стандартном режиме значение параметра `\tolerance` равно 200. Действие команды `\sloppy` сводится в основном к тому, что она устанавливает значение `\tolerance` равным 10000, то есть максимально возможному. Это объясняет, почему в режиме, определенном этой командой, \TeX создает сколь угодно разреженные строки. При этом, так как \TeX не различает строки с мерой разреженности, большей 10000, может получиться так, что одна из строк абзаца окажется совершенно ужасной: \TeX вложит в нее «всю разреженность», чтобы не увеличивать численный показатель «плохости абзаца» (это — именно то число, которое \TeX минимизирует при переборе различных вариантов верстки данного абзаца; грубо говоря, оно тем больше, чем больше разреженных строк). Поэтому разумным решением во многих случаях будет увеличить значение `\tolerance`, но не до максимума, как это делает команда `\sloppy`, а до более разумной величины (скажем, 300 или 400). После этого \TeX , с одной стороны, получит большую свободу при верстке абзаца, а с другой — не сможет

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

уже создавать абзацы, в которых все строки, кроме одной, приемлемы, а одна разрежена до безобразия.

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`).

Г. Как менять длину абзаца. Иногда абзац не помещается на страницу из-за того, что он на строчку-другую длиннее, чем нужно. В этом случае можно попросить Т_ЭХ сделать его короче на одну или две строчки. Для этого надо написать

```
\looseness=-1
```

— и Т_ЭХ будет стараться, чтобы абзацы занимали на одну строчку меньше, чем при оптимальной верстке. Если абзац короткий (скажем, занимает всего две строчки), то из этого, конечно, ничего не получится. Если же абзац достаточно длинный, то гибкости Т_ЭХовского алгоритма верстки хватит на то, чтобы эта цель была достигнута. Нет смысла писать такую команду в преамбуле (вряд ли нам нужно, чтобы все до единого абзацы были на одну строчку короче, чем при оптимальной верстке), поэтому лучше дать эту команду внутри группы, чтобы после окончания этой группы восстановился прежний режим. Как обычно, в группу должна входить и пустая строка, завершающая абзац, чтобы к моменту верстки значение, присвоенное нами параметру `\looseness`, не забылось.

Можно присвоить параметру `\looseness` и значение -2 ; в этом случае Т_ЭХ будет стараться делать абзацы короче на две строки (если не выйдет, то хоть на одну, а если и это не выходит, то оставит все как есть). Можно также присваивать `\looseness` положительные значения — в этом случае Т_ЭХ будет стараться делать абзацы, которые содержат *больше* строчек, нежели оптимальные. По умолчанию значение `\looseness` равно нулю.

Д. Дополнительные тонкости с переносами. Вы можете влиять на частоту переносов в абзацах, сверстанных Т_ЭХом. Для этой цели предназначен параметр `\hyphenpenalty`. По умолчанию его значение равно 50. Можно присвоить этому параметру бóльшее значение, и тогда переносов будет меньше. Точнее говоря: если у Т_ЭХа будет возможность выбирать, сделать лишний перенос или же обойтись без него, растянув строку чуть больше,⁶ то Т_ЭХ будет склоняться ко второй из этих альтернатив 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 будут вообще запрещены. Такой режим верстки разумно, например, установить для абзацев, написанных на языке, для которого в Вашей реализации Т_ЭХа нет таблицы переносов, чтобы Т_ЭХ не сделал переносов во французском тексте по английским правилам.

Наряду с `\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. Напомним, что в таких словах автоматический перенос возможен только в том месте, где дефис делит слово

⁶Не превышая значения `\tolerance`, разумеется!

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 вышеописанных 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`, чтобы он смог побольше растягивать строки.

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 большой 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 заглавной буквы, присвойте параметру `\uchyph` значение 1.

7. 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.

7.1. Понятие о режимах \TeX а

В процессе обработки исходного текста \TeX в каждый момент находится в одном из трех режимов: горизонтальном, вертикальном или математическом. Несколько упрощая ситуацию, дело можно представить так:

- В процессе обработки текста (от появления первой же буквы до команды «закончить абзац», например, пустой строки) \TeX находится в горизонтальном режиме.
- Между абзацами, а также в начале работы (например, в процессе обработки преамбулы к \LaTeX овскому файлу) \TeX находится в вертикальном режиме.
- При обработке математических формул (см. главу II) \TeX находится в математическом режиме.

В вертикальном режиме все пробелы и пустые строки игнорируются, так что между пустой строкой, завершающей абзац, и новой порцией собственно текста незачем заботиться о лишних или недостающих пробелах (ср. стр. 12).

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 уже совсем другой абзац.

Это --- абзац, который мы не намерены завершать пустой строкой, как раньше. `\par` А это уже совсем другой абзац.

Иногда для ясности, когда в исходном файле присутствует сложная комбинация из \TeX овских команд, имеет смысл обозначить конец абзаца именно таким способом.

Идущие подряд несколько команд \par , команда \par , за или перед которой следует пустая строка, и т. п. — все это равносильно одной пустой строке или одной команде \par (точно так же, как несколько пустых строк равносильны одной); дополнительный промежуток между абзацами Вы таким образом не создадите. В разделе 7.3 рассказано, 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.

7.2. 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):

Этот абзац будет сверстан без отступа.

В этом абзаце отступ будет присутствовать.

\noindent Этот абзац будет сверстан без отступа.

В этом абзаце отступ будет \noindent присутствовать.

Команда \noindent действует только на тот абзац, который с нее начинается; если ее поместить внутри абзаца, то вообще ничего не произойдет (что и иллюстрирует второй из абзацев в нашем примере). Стало быть, между \noindent и абзацем, к которому она относится, не должно быть пустой строки (иначе получится, что \noindent относится к «пустому абзацу», заканчивающемуся этой пустой строкой).

В большинстве случаев, когда разумно сделать абзац без отступа, \LaTeX заботится об этом сам, так что Вам не придется пользоваться командой \noindent чересчур часто.

Пользуясь понятием режима, можно сказать так: в вертикальном режиме команда \noindent означает «начать новый абзац без абзацного отступа», а в горизонтальном (и математическом, коль на то пошло) режиме она означает «ничего не делать».

7.3. Вертикальные промежутки

Большинство вертикальных промежутков (например, между заголовком раздела и его текстом) \LaTeX устанавливает самостоятельно, и Вам об этом можно не заботиться. Иногда возникает необходимость сделать дополнительный вертикальный промежуток между абзацами. Подобно тому, как внутри абзацев для задания промежутков вручную разумнее пользоваться не командами, явно задающими размер промежутка, а командами вроде $\,$, или \quad , так и для задания промежутков между абзацами в первую очередь полезны 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, 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 перечисленными командами, зависит от стиля документа. Эти размеры совпадают со значениями параметров `\smallskipamount...` `\bigskipamount`, о которых шла речь на стр. 74.

Если Вы хотите задать размер вертикального промежутка в явном виде, можно воспользоваться командой `\vspace`. Подобно команде `\hspace` (см. стр. 62), 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.

У команды `\vspace` есть вариант со звездочкой после имени команды. Если написать, допустим, `\vspace*{1cm}`, то будет создан вертикальный промежуток в 1 сантиметр, не пропадающий даже в том случае, если в этом месте произойдет 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_ЭХовских единицах, а `plus` и `minus` — так называемые «ключевые слова» Т_ЭХа (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 (если, например, мы разрешили Т_ЭХу 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, насколько, самое большее, можно растянуть интервал, в то время как z указывает, насколько, самое большее, можно его ужать. Говоря Т_ЭХническим языком, команда `\vspace` вставляет в страницу «клей»; расстояния, указанные после `plus` и `minus`, называются соответственно `plus` и `minus`-компонентами этого клея. Если `plus`- или `minus`-компонента в аргументе команды `\vspace` не указана, то соответствующий интервал не сможет растягиваться (сжиматься). Большинство вертикальных интервалов, автоматически вставляемых Л_АТ_ЭХом, обладают растяжимостью и/или сжимаемостью, что помогает при нахождении оптимальных разрывов страниц.

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 (самое бóльшее — на 2 мм) или сжиматься (самое бóльшее — на 1 em). В простых приложениях такие конструкции, как правило, не встречаются. Мы еще будем говорить о них в разделе 3.3 главы VIII.

7.4. Интерлиньяж

В полиграфии этим красивым словом называется интервал между строками. ЛАТЭХовские команды наподобие `\small`, устанавливающие размер шрифта, автоматически устанавливают и размер интервала между строками, и вручную менять его не следует (потому мы и не рассказываем, как это делать; любопытствующий читатель может узнать подробности в книге [2]). Можно, однако (и иногда это бывает необходимо), *пропорционально* увеличить или уменьшить все интервалы между строками: например, для того, чтобы подогнать число страниц в документе к требуемому. Если, скажем, Вы хотите увеличить интервалы между строками на 10%, то есть в 1,1 раза, то следует написать так:

```
\renewcommand{\baselinestretch}{1.1}
```

Вместо десятичной точки можно использовать и десятичную запятую. Если эта команда дана внутри группы, то по выходе из группы ее действие прекращается (если Вы хотите, чтоб измененный интерлиньяж присутствовал в каком-то абзаце, то, как и в случае с командами, меняющими размер шрифта, группа должна кончаться *после* конца абзаца). Если вышеописанная команда будет дана в преамбуле, то, естественно, ее действие распространится на весь документ.

Между абзацами можно организовать дополнительные вертикальные интервалы. Именно, в ТЭХе есть параметр `\parskip` со значением длины; если присвоить ему ненулевое значение, например, написав

```
\parskip=3mm
```

то между абзацами будет делаться отступ в 3 мм (в дополнение к обычному межстрочному интервалу). В обычных случаях нет смысла присваивать параметру `\parskip` новое значение, поскольку оно вполне разумно устанавливается в стандартных ЛАТЭХовских стилях.

На самом деле в стандартных стилях `\parskip` является *растяжимой* длиной (см. стр. 80). Именно, естественный размер `\parskip`'а 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 (максимум на один пункт). При желании можно изменить как естественный размер, так и растяжимую компоненту `\parskip`'а с помощью команды `\setlength`, о которой пойдет речь в разделе VII.3. Будем надеяться, что Вы будете делать это сознательно.

7.5. Управление разрывами страниц

Как Вы могли убедиться из раздела, посвященного абзацам, ТЭХ 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 ТЭХ 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 «одноходовый». Как только \TeX набирает достаточно строк, чтоб заполнить страницу, 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.

А. Запрет разрыва страницы. Чтобы запретить разрыв страницы, используется команда `\nopropagebreak`. Если поставить ее после конца абзаца, то разрыв страницы после этого абзаца будет запрещен. Если после конца абзаца присутствуют совместно как команда `\nopropagebreak`, так и команда для дополнительных вертикальных промежутков, то `\nopropagebreak` должна идти первой, в противном случае она не подействует.

Команда `\nopropagebreak` может принимать необязательный аргумент — целое число от 0 до 4. Если она дана с этим аргументом, то 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 `\nopropagebreak[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.

Наряду с командами, запрещающими разрыв страниц в указанном месте, \LaTeX предоставляет способ «глобально» затруднить \TeX у разрывы страниц. Для этого служит команда `\samespage`. После этой команды разрывы страниц станут возможны только между абзацами, но не внутри абзацев и не между текстом и выключной формулой. Если дать команду `\samespage` внутри группы, то после конца группы действие этой команды прекращается (потому что это действие сводится к изменению некоторых не рассмотренных нами параметров).

Б. 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 Вы пользуетесь 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. раздел IV.5), то перед выдачей новой страницы они будут напечатаны.

Команда `\cleardoublepage` делает все то же, что и `\clearpage`, но при этом в некоторых стилях (в тех, которые предусматривают разные поля для страниц с четным и нечетным номером — см. в разделе IV.1 по поводу стилевой опции `twoside`) новая страница обязательно имеет нечетный номер (если необходимо, при этом создается дополнительная пустая страница).

Все вышеперечисленные команды для разрыва страницы действуют даже в том случае, если команда `\samespage` ранее его запретила.

Если поставить подряд две команды `\newpage` (или `\clearpage`), то в печатном тексте чистая страница *не* получится. Чтобы создать чистую страницу, надо \LaTeX немного «обмануть»: в промежутке между двумя командами для разрыва страницы дать команду `\mbox{}`.

Наконец, существует команда `\pagebreak`, формально аналогичная команде `\linebreak` (см. стр. 73). 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). Разумеется, вид у такой страницы еще хуже, чем у «укороченной» страницы с нормальными интервалами. Если дать команду `\pagebreak` с необязательным аргументом (целым числом от 0 до 4), то этот аргумент будет выражать степень желательности разрыва страницы в данном месте: если 0, то это всего лишь разрешение разорвать страницу, если 4, то разрыв обязателен, в остальных случаях степень желательности растет с ростом аргумента от 1 до 3.

Каждую из вышеперечисленных 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.

7.6. Набор в две колонки

Если Вам необходимо набирать в две колонки весь документ, то это надо сделать, указав в команде `\documentstyle` соответствующую «стилевую опцию» (см. раздел IV.1). Если же в две колонки надо набрать не весь текст, а только его часть, к вашим услугам команда `\twocolumn`. Действует она так: сначала выполняется команда `\clearpage`, а затем с новой страницы, созданной этой командой, начинается набор в две колонки.

Иногда бывает необходимо сделать так: начать новую страницу, в начале этой 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 `\onescolumn` осуществляет переход от двухколонного набора к одноколонному (предварительно она опять-таки выполняет команду `\clearpage`).

7.7. Заключительные замечания о разрывах страниц и вертикальных интервалах.

Мы уже отмечали, что \TeX овские алгоритмы создания страниц не обладают той же гибкостью, что алгоритм разбиения абзаца на строки. Поэтому не надо слишком увлекаться принудительными разрывами и запретами разрывов страниц и командами наподобие `\vspace*`. Даже такая замечательная программа, как \TeX , не сможет удовлетворить логически противоречивым требованиям; если ограничений на разрывы страниц слишком много, то \TeX будет вынужден сделать эти разрывы, исходя из формального смысла своих алгоритмов. При этом, скорее всего, на печати Вы получите много страниц, разорванных в самых неожиданных и неудачных с точки зрения человека местах, а на экране — много сообщений вроде такого:

```
Underfull \vbox (badness 10000)
has occurred while \output is active
```

Если Вы регулярно сталкиваетесь с такими неприятностями, имеет смысл заново продумать принципы организации Вашего текста, а может быть, и кое-что переизложить. Избавить-

ся от растянутых по вертикали страниц можно, если дать в преамбуле документа команду `\raggedbottom`, разрешающую делать страницы неодинаковой высоты (некоторые стили дают эту команду автоматически, но даже если окажется, что Вы ее продублировали, ничего плохого не случится). Действие, противоположное `\raggedbottom`, вызывается командой `\flushbottom`.

Наконец, если трудности возникают оттого, что Вы часто оставляете в тексте место командой `\vspace*` (например, чтобы вклеить рисунок), то Вам стоит воспользоваться «плавающими» иллюстрациями (см. раздел IV.5).

8. Специальные абзацы

В разделе, посвященном абзацам, уже упоминалось о том, как можно верстать текст без выравнивания по правому краю. Сейчас речь пойдет о других подобных случаях, когда требуется специальная верстка текста. Большинство описываемых в этом разделе способов верстки реализованы в виде окружений (см. стр. 15); не забывайте, что всякое окружение ограничивает группу, так что если Вам нужно будет не только получить текст специального вида, но и переключить шрифт, Вы можете дать команду переключения шрифта внутри окружения и не заботиться специально о восстановлении прежнего шрифта: вместе с окружением кончится и группа, и прежний шрифт восстановится автоматически.

8.1. 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. Пример:

Типографское правило гласит, что	Типографское правило гласит, что <code>\begin{quote}</code>
никакая строка не должна быть длиннее 66 символов	никакая строка не должна быть длиннее 66 символов <code>\end{quote}</code>
Поэтому текст в газетах набира- ется в несколько колонок.	Поэтому текст в газетах набирается в несколько колонок.

Как Вы можете заметить из этого примера, текст, оформленный окружением `quote`, не имеет абзацного отступа и отделяется от окружающего текста пробелами. 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. стр. 14).

Для длинных цитат, состоящих из нескольких абзацев, удобнее использовать окружение `quotation`. Оно полностью аналогично `quote`, за тем исключением, что в тексте, оформленном этим окружением, делается абзацный отступ.

8.2. Центрирование, прижатие текста к краю

Для этих целей используются окружения `center` (для центрирования), а также `flushleft` и `flushright` (для верстки текста, прижатого к 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 прижата к левому или правому краю (для окружений `flushleft` и `flushright` соответственно).

Окружение `flushleft` по своему действию практически эквивалентно команде `\raggedright` (см. стр. 74); 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 прижата слева или справа; для этого достаточно после каждой строки, которую Вы хотите центрировать (прижать справа...), поставить команду `\` (см. стр. 73); следующая строка будет принадлежать тому же абзацу, и, стало быть, опять же будет центрирована (прижата). А если Вы не будете делать разбиения на строки командой `\`, то это будет сделано автоматически, с появлением вышеописанных ненормальных абзацев (на стр. 16 приведен пример того, что может в этом случае получиться). Вот примеры разумного использования этих окружений:

левый марш	<code>\begin{flushleft}</code> левый <code>\</code> марш <code>\end{flushleft}</code>
наше дело правое	<code>\begin{flushright}</code> наше дело <code>\</code> правое <code>\end{flushright}</code>
а мы всегда в центре	<code>\begin{center}</code> а <code>\</code> мы <code>\</code> всегда <code>\</code> 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 ЛАТЭХом с обычным абзацным отступом, в противном случае — без отступа.

Нередко возникает необходимость создать одну-единственную центрированную строку. Наряду с окружением `center` для этих целей можно использовать команду `\centerline`, что экономит немало нажатий на клавиши:

Пишем, пишем текст...	Пишем, пишем текст <code>\dots</code>
Центрированная строка	<code>\medskip</code> <code>\centerline{\sf Центрированная</code> <code>строка}</code> <code>\medskip</code>
И продолжаем благополучно писать дальше и дальше.	И продолжаем благополучно писать дальше и дальше.

Обратите внимание, что аргумент команды `\centerline` расположен внутри группы (так что, например, можно внутри ограничивающих его фигурных скобок сменить шрифт, а в дальнейшем тексте он автоматически восстановится). В отличие от окружения `center` эта команда не создает автоматически дополнительных вертикальных пробелов вокруг центрированного текста, так что при необходимости их надо создавать вручную (как в вышеприведенном примере).

8.3. Стихи

В принципе вполне можно набирать стихи с помощью окружений `flushleft` или `center`, разделяя строки командой `\\`, а строфы, например, командой `\\[4pt]` (см. стр. 74 по поводу этих команд). Кроме того, в \LaTeX для набора стихов предусмотрено специальное окружение `verse`. Строки в нем разделяются командой `\\`, а строфы — пустой строкой. При этом строки получаются выровненными по левому краю и отодвинутыми от левой границы текста. Если строка окажется слишком длинной, она будет перенесена на следующую строку (не исключено, что в каких-то словах будет сделан перенос) и сдвинута примерно на 15 пунктов вправо. Пример:

Здесь любит медведь	<code>\begin{verse}</code>
Иногда посидеть	Здесь любит
И подумать: “А чем	медведь\\
бы такое заняться-	Иногда посидеть\\
ся?”	И подумать:
	‘‘А чем бы такое заняться?’’
	<code>\end{verse}</code>

Создатель \LaTeX а Лесли Лэмпорт предсказывал, что окружение `verse` будет обругано поэтами.

Наличие или отсутствие абзацного отступа в абзаце после окружения `verse` определяется по тем же правилам, что для `quote` и `quotation`.

8.4. Перечни

Для печати перечней используются окружения `itemize`, `enumerate` и `description`:

- Окружение `itemize` предназначено для простейших перечней.
- Окружение `enumerate` предназначено для нумерованных перечней.
- Окружение `description` предназначено 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 параметром). Разберем последовательно, как работают указанные окружения.

А. Простейшие перечни (`itemize`). Каждый элемент перечня вводится командой `\item` без параметра.

- На печати каждый элемент перечня снабжается черным 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 L^AT_EX выдаст сообщение об ошибке.

Вот как выглядел в исходном файле предшествующий текст:

```
\begin{itemize}
\item На печати каждый...
\item Перечни могут быть
вложенными друг в друга:
  \begin{itemize}
  \item Максимальная глубина вложенности равна 4.
  \item Отступы и символы перед элементами
        выбираются автоматически.
  \end{itemize}
\item На втором уровне элементы...
\item При попытке вложить...
\end{itemize}
```

Внутри окружения `itemize` до первой команды `\item` не должно идти никакого текста или же команд, генерирующих текст. Если Вы попытаетесь проигнорировать этот запрет, то L^AT_EX выдаст Вам сообщение об ошибке. Другие команды (например, команды смены шрифта) могут идти и до первого `\item`'а.

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 фигурных 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:

<ul style="list-style-type: none"> • Этот элемент перечня помечен стандартно. <p>Один Здесь мы сами задали заголовок.</p> <p>Два Здесь тоже.</p>	<pre>\begin{itemize} \item Этот элемент перечня помечен стандартно. \item[\sf Один] Здесь мы сами задали заголовок. \item[Два] Здесь тоже. \end{itemize}</pre>
---	--

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, если Вы хотите, чтобы заголовки

элементов перечня были выровнены по левому, а не по правому краю, лучше пользоваться окружением `description`, о котором речь пойдет ниже.

Если первый отличный от пробела символ после команды `\item` является открывающей квадратной скобкой, то ЛАТЭХ 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...
```

Б. Нумерованные перечни (`enumerate`). В таких перечнях каждый элемент также вводится командой `\item` без параметра, но на печати он будет отмечен не значком, а номером (эти номера создаются ЛАТЭХом 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 ЛАТЭХ выдаст сообщение об ошибке.

В исходном тексте это выглядело так:

```
\begin{enumerate}
\item В окружении {\tt enumerate}
элементы списка нумеруются цифрами или буквами.
\item Нумерация производится
автоматически.
\item Перечни могут быть
вложенными друг в друга:
  \begin{enumerate}
  \item Максимальная глубина...
  \item Отступы и обозначения...
  \end{enumerate}
\item На втором уровне...
...
\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. стр. 17). 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 редактирования текста переставите какие-то пункты или добавите новые, то номер пункта 3 может измениться. Вместо того, чтобы каждый раз отсчитывать, которым по счету идет этот пункт, можно пометить элемент перечня с помощью команды `\label` (см. раздел I.2.11). Команду `\label` удобно ставить сразу после команды `\item`, вводящей помечаемый элемент перечня, но в принципе можно поставить ее и позже — до следующего `\item`'а.

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.	<code>\begin{enumerate}</code>
	<code>\item Переходите улицу только на зеленый свет.</code>
2. Стоящий трамвай обходить можно, а другие виды транспорта — нет.	<code>\item \label{tram}</code>
	Стоящий трамвай обходить можно, а другие виды транспорта --- нет.
	<code>\end{enumerate}</code>
Согласно правилу 2, сформулированному на странице 89, обходить стоящий автобус нельзя.	Согласно правилу~\ref{tram}, сформулированному на странице~\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` также может иметь необязательный аргумент, который работает так же, как в окружении `itemize`. Соответственно, остается в силе и сделанное в предыдущем подпункте предупреждение относительно того, что будет, если первый отличный от пробела символ после `\item` является открывающей квадратной скобкой.

В. 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. стр. 15), представляющим собой этот заголовок. Следующий пример получен в результате переноса аналогичного примера из книги [1] на отечественную почву:

Многие наши руководители прославились интересными высказываниями:

Ю.В. Андропов: Политика — это не игра в покер, где, продувшись, можно отыгратья.

К.У. Черненко: Чтобы лучше жить, надо лучше работать.

М.С. Горбачев: Процесс пошел.

Многие наши руководители прославились интересными высказываниями:

```
\begin{description}
\item[Ю.В.~Андропов:]
Политика --- это не игра
в покер, где, продувшись,
можно отыгратья.
\item[К.У.~Черненко:]
Чтобы лучше жить, надо
лучше работать.
\item[М.С.~Горбачев:]
Процесс пошел.
\end{description}
```

Как Вы могли заметить, заголовки элементов перечня оформляются в окружении `description` полужирным шрифтом. Если Вас не устраивает этот шрифт, можно аргумент команды `\item` начать с команды переключения шрифта, скажем, `\rm` или `\sl`.

Внутри окружения `description` до первой команды `\item` не должно идти никакого текста или же команд, генерирующих текст.

Если в заголовке элемента перечня присутствует закрывающая квадратная скобка, то \LaTeX решит, что именно на ней заканчивается необязательный аргумент команды `\item`, в результате чего на печати получится совсем не то, что Вы хотели. Чтобы избежать этой неприятности, надо эту квадратную скобку (либо, что еще проще, весь заголовок) заключить дополнительно в фигурные скобки (внутри квадратных).

Г. Другие виды перечней. Если Вас не устраивает навязываемое Вам \LaTeX ом оформление перечней (например, вид пометок, которыми отмечаются элементы перечня `itemize`), можно это оформление изменить. Как это сделать, будет рассказано в разделе VII.2.5; это несложно. Несколько труднее, к сожалению, сделать так, чтобы буквы, которыми нумеруются элементы нумерованного перечня, были русскими, а не латинскими (как в примерах в этой книге). Желательно, чтобы в устанавливаемой у Вас русификации \LaTeX а были определены соответствующие команды. \LaTeX позволяет менять оформление перечней и более серьезным образом, создавая перечни иного типа, чем рассмотренные выше. На данный момент у нас еще не хватает \TeX ники для того, чтобы освоить эти возможности \LaTeX а, но в главе IX будет рассказано и об этом.

8.5. Буквальное воспроизведение (`verbatim`, `verb`)

Окружение `verbatim` предназначено для воспроизведения текста шрифтом `typewriter` в точности так, как он выглядит в файле. Одной только команды `\tt` для этого недостаточно, поскольку воспроизводимый текст может содержать, например, команды \TeX а, и необходимо, чтобы они печатались, но не исполнялись.

Между `\begin{verbatim}` и `\end{verbatim}` могут идти любые символы, за исключением последовательности символов `\end{verbatim}` (в том числе, например, `\` или не имеющие пар фигурные скобки). В строке `\end{verbatim}`, завершающей окружение `verbatim`, *не* должно быть пробела между `\end` и `{verbatim}` (в изъятие из общего правила: обычно такой пробел, как и вообще пробел после имени команды, состоящего из букв, безвреден).

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 — тот символ, что стоял непосредственно после `\verb`. После `\verb` не должно быть пробела. Пример:

Команда `\dots` задает многоточие. Знак " в `TeX`е используется редко.

Команда `\verb"\dots"` задает многоточие. Знак `"` в `TeX`е используется редко.

Описанные окружение и команда удобны, когда надо имитировать машинописный текст, текст на мониторе компьютера, или набирать тексты компьютерных программ. В данном руководстве `\verb` и `verbatim` широко использовались для набора `LaTeX`овских и `TeX`овских команд.

У команды `\verb` и окружения `verbatim` есть варианты «со звездочкой» (см. стр 16). 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:

```
{\tt \symbol{"5C}\sqrt}
```

Если Вы забудете «закрывающий символ» в команде `\verb` или сделаете опечатку в тексте `\end{verbatim}`, то в лучшем случае получите уйму сообщений об ошибке, а в худшем — приведете `TeX` в такое состояние, что компьютер придется перезагружать.

8.6. Абзацы нестандартной формы

Пусть нам потребовалось создать абзац с «отрицательным» абзацным отступом, в котором все строки, кроме первой, начинаются на расстоянии 1 см от полей. Этого можно добиться следующим образом:

С помощью `TeX`а нетрудно создавать абзацы нестандартной формы. Простейший пример — то, что по-английски называется `hanging indentation`.

```
\hangindent=1cm \noindent
С помощью \TeX{}а нетрудно
создавать абзацы
нестандартной формы.
Простейший пример --- то, что
по-английски называется
hanging indentation.
```

Здесь `TeX`овский параметр `\hangindent` означает величину отступа от полей во всех строках абзаца, кроме первой (по умолчанию значение этого параметра равно нулю). Обратите внимание, что мы начали абзац командой `\noindent`, чтобы первая строка не началась с абзацным отступом.

Пусть теперь нам хочется, чтобы дополнительный отступ, величина которого задана параметром `\hangindent`, начинался не со второй строки, а, скажем, с третьей. Для этого нам надо установить еще один `TeX`овский параметр, обозначаемый `\hangafter`:

Следующая проблема — сделать так, чтобы дополнительный отступ начинался не со второй строки, а позже. Как видите, добиться этого тоже несложно.

```
\hangindent=1cm \hangafter=2
\noindent
```

Следующая проблема --- сделать так, чтобы дополнительный отступ начинался не со второй строки, а позже. Как видите, добиться этого тоже несложно.

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, ... $|n|$. Пример:

С помощью рассмотренных нами средств `TeX` можно выкапывать в абзацах небольшие ямки. На место образующегося белого прямоугольника можно, например, наклеить иллюстрацию.

```
\hangindent=1.5cm \hangafter=-3
\noindent
```

С помощью рассмотренных нами средств `TeX` можно выкапывать в абзацах небольшие ямки. На место образующегося белого прямоугольника можно, например, наклеить иллюстрацию.

Можно сделать так, что дополнительный отступ, задаваемый параметром `\hangindent`, будет делаться не слева, а справа. Для этого значение этого параметра надо сделать отрицательным. Именно, если значение `\hangindent` равно $h < 0$, то дополнительный отступ размером $|h|$ будет отсчитываться от *правого*, а не левого поля (в каких именно строках будет этот дополнительный отступ, по-прежнему определяется значением `\hangafter`):

На сей раз нам захотелось приклеить картинку не слева, а справа. Что ж, `TeX` позволяет сделать и так, было бы желание. Вскоре Вы увидите, что и это — не предел.

```
\hangindent=-2cm \hangafter=2
\noindent
```

На сей раз нам захотелось приклеить картинку не слева, а справа. Что ж, `TeX` позволяет сделать и так, было бы желание. Вскоре Вы увидите, что и это --- не предел.

После каждой команды «завершить абзац» (иными словами, после каждой пустой строки или команды `\par`) значения параметров `\hangindent` и `\hangafter` возвращаются к принятым по умолчанию. Отметим еще, что было бы неразумно играть с этими параметрами внутри `LaTeX`овских окружений наподобие `itemize` или `quote`.

Если Вам не хватает возможностей, которые дает варьирование параметров `\hangindent` и `\hangafter`, то вот Вам пример, как с помощью Т_ЭХа создать абзац совсем уж причудливой формы. Все переносы в словах и места разрывов строк были найдены Т_ЭХом автоматически.

Начало этого причудливого абзаца выглядело в исходном тексте так:

```
\parshape=7
0cm 4cm 0.5cm 5cm 1cm 6cm 1.5cm 7cm
2cm 6.5cm 1.8cm 6cm 1.7cm 5cm
\noindent \small
Если Вам не хватает возможностей...
```

Смысл этих команд следующий. Число 7, 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, 4cm — ее длина, 0.5cm — отступ второй строки от левого поля, 5cm — 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 Т_ЭХом; если же строк, как в нашем примере, 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 не командой `\noindent`, а обычным образом, 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 Т_ЭХ считает, что эта формула занимает три строки, причем сама формула расположена в средней из этих трех (реально формула может, разумеется, занимать больше или меньше места).

9. Линейки

9.1. 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 Т_ЭХе линейкой (*rule* по-английски) называется любой черный прямоугольник. Для создания линеек в Л^AТ_ЭХе используется команда `\rule`. У этой команды два обязательных аргумента: первый задает ширину прямоугольника-линейки, второй — высоту (оба этих размера должны быть заданы в используемых Т_ЭХом единицах измерения — см. стр. 16). Линейка, созданная командой `\rule`, рассматривается Т_ЭХом 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 `\rule{.5em}{15pt}`, а после нее продолжится обычный текст. Сравните также `\rule{.5em}{15pt}` и `\rule[-3pt]{.5em}{15pt}`.

В этом месте, прямо посреди абзаца, будет линейка `\rule{.5em}{15pt}`, а после нее продолжится обычный текст. Сравните также `\rule{.5em}{15pt}` и `\rule[-3pt]{.5em}{15pt}`.

9.2. Т_ЭХовские команды для генерации линеек

Т_ЭХовская команда `\rule` обладает рядом недостатков. Например, то обстоятельство, что создаваемые с ее помощью линейки воспринимаются Т_ЭХом как буквы, усложняют такую операцию, как печать линейки, простирающейся во всю ширину страницы (если между абзацами, то есть в «вертикальном режиме», сказать что-нибудь наподобие

```
\rule{10cm}{1pt}
```

то линейка начнется не с левого края текста, а после абзацкого отступа: Т_ЭХ решит, что с этой «буквы» начинается новый абзац); кроме того, при печати линеек с помощью команды `\rule` необходимо заранее знать их длину и ширину, что не всегда удобно (например, если линейка должна идти во всю ширину текста, то надо точно знать, чему эта ширина равна, либо, по крайней мере, знать, как она обозначается в Т_ЭХе). Избавиться от этого неудобства можно с помощью Т_ЭХовских 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 `y`, 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 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 backslash'a перед `height` не является опечаткой (`height` — не команда, а одно из так называемых «ключевых слов» Т_EXа, наподобие уже встретившихся нам в разделе 7.3 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 Л^AT_EXовской 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 Т_EXом как ключевое, что приведет к сообщению об ошибке. При верстке текста на русском языке 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».

9.3. Невидимые линейки

Высота и/или ширина линейки может быть нулевой или отрицательной. Линейки отрицательной 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 Т_EXе и Л^AT_EXе для такой линейки предусмотрена специальная команда `\strut`. Невидимая линейка, создаваемая этой командой, имеет нулевую ширину; высота же ее установлена автором Л^AT_EXа 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 шрифте, которым набрана эта книга, команда `\strut` создает линейку ширины 0, поднимающуюся над уровнем строки на 10.14995pt и опускающуюся ниже уровня строки на 4.35004pt.

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 целых три слова подряд. Выглядит это не очень удачно: в словах с буквами вроде `p`, 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`:

<u>целых</u> <u>три</u> <u>слова</u>	<code>\underline{целых\strut}</code>
	<code>\underline{три\strut}</code>
	<code>\underline{\strut слова}</code>

Как видите, `\strut` можно ставить хоть после слова, хоть перед ним (и даже посередине, если Вы не запутаетесь с пробелами). Вся функция 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 главе II рассказывается про команду `\mathstrut`, выполняющую аналогичные функции в математических формулах.

Другие примеры использования невидимых линеек читатель найдет в главе, посвященной верстке с выравниванием; в главе о блоках мы также встретимся с линейками.

Глава IV.

Оформление текста в целом

В этой главе мы рассмотрим такие вопросы, как общий стиль оформления документа, разбиение текста на разделы, титульный лист, оглавление и пр. ЛАТЭХ освобождает Вас от многих забот об оформлении документа, но при этом и навязывает такие черты оформления, которые могут Вам не нравиться. От этого «диктата» можно отчасти отойти, если модифицировать стандартные стили (в последней главе мы расскажем, как это можно делать). В принципе можно создать и свой собственный стиль, весьма далекий от стандартных, но для этого требуются более глубокие познания в ТЭХе, чем дает эта книга. Начнем же мы с того, что систематически рассмотрим вопрос о стандартных стилях.

1. Стили

Команда `\documentstyle`, с которой начинается любой ЛАТЭХовский файл, имеет один обязательный аргумент — название основного стиля — и один необязательный, размещающийся перед обязательным — список «стилевых опций» (см. стр 12). Основной аргумент — это название «основного стиля» оформления документа. В первой главе (см. раздел I.2.4) мы уже перечислили четыре основных стиля, предоставляемых ЛАТЭХом: `article`, `report`, `book` и `letter`. Оставляя последний в стороне, рассмотрим подробнее три основных стандартных стиля, предоставляемых нам ЛАТЭХом.

Стиль `article` удобно применять для статей, стиль `report` — для более крупных статей, разбитых на главы, или небольших книг, стиль `book` — для книг. В нижеследующей таблице перечислены некоторые черты оформления, присущие стандартным стилям. В ней знак «+» означает «всегда присутствует», знак «-» означает «всегда отсутствует», знак « \mp » означает «по умолчанию отсутствует, но будет присутствовать, если задать стилевую опцию или специальную команду», знак « \pm » означает «по умолчанию присутствует, но можно отменить с помощью специальной команды».

	article	report	book
Автоматически нумерующиеся разделы	+	+	+
Разбиение на главы	—	+	+
Разбиение на части	+	+	+
«Двусторонняя»* печать	∓	∓	+
Титульный лист	∓	+	+
Колонтитулы	∓	±	±
Высота всех страниц одинакова	∓	∓	±
Набор в две колонки	∓	∓	∓

* с разными полями для четных и нечетных страниц.

Мы не стремились охватить в этой таблице все детали отличий между стандартными стилями (например, колонтитулы в трех стилях оформляются немного по-разному).

Опишем теперь стилевые опции. Напомним (см. стр. 12), что список стилевых опций через запятую ставится в квадратных скобках перед основным аргументом команды `\documentstyle`. Самые важные и часто употребляемые стилевые 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.

Стилевая опция `twoside` задает печать с разными полями на нечетных и четных страницах (как в книгах). Ее можно указывать только для стилей `article` и `report`; в стиле `book` такая печать получается автоматически, и указывать опцию `twoside` при этом основном стиле нельзя.

Стилевую опцию `twocolumn` можно задавать для любого из трех основных стилей. 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. стр. 76), иначе будет получаться много строк, выбивающихся за колонку.

Стилевая опция `titlepage` применима только в том случае, если основной стиль — `article`. Если она задана, то у документа будет напечатан титульный лист (в стилях `report` и `book` титульный лист, как было указано выше, делается всегда).

Опция `draft` пригодна для любого из трех стилей. Если она включена, то каждая выбивающаяся на поля строка (то есть строка, о которой выдается сообщение «`Overfull \hbox`» — см. стр. 69), 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. раздел III.7.7). По умолчанию такой режим устанавливается стилями `article` и `report`, если только в качестве стилевой опции не указана «двусторонняя» печать (стилевая опция `twoside`). Во всех остальных случаях \LaTeX будет, по умолчанию, делать все страницы одинаковой высоты.

Следующие две стилевые опции, применимые к любому из основных стилей, влияют на оформление выключных математических формул; если Вы пропустили при чтении соответствующие места в главе II, то пропустите и это место. Опция `fleqn` означает, что выключные формулы, заданные с помощью окружений `equation`, `eqnarray` и `displaymath`, а также пары команд `\[` и `\]`, будут напечатаны не в центре строки, а в ее левой части. Опция `leqno` означает, что номера формул, генерируемые окружениями `equation` и `eqnarray`, будут печататься не справа, а слева.

Титульному листу и разбиению документа на разделы будут посвящены отдельные разделы; чтобы завершить наш обзор различных стандартных вариантов стиля, нам осталось обсудить колонтитулы и номера страниц. Этому посвящен следующий пункт.

1.1. Стиль оформления страницы

Колонтитулы являются частным случаем более общего вопроса об оформлении страницы. Для задания стиля оформления страницы в Л^AT_EXе предусмотрена команда `\pagestyle`. Эта команда имеет один обязательный аргумент — слово, обозначающее этот стиль. При использовании стандартными стилями Л^AT_EXа это слово должно быть одним из следующих:

`empty` Нет ни колонтитулов, ни номеров страниц.

`plain` Номера страниц ставятся внизу в середине строки, колонтитулов нет.

`headings` Присутствуют колонтитулы (включающие в себя и номера страниц).

`myheadings` Присутствуют колонтитулы, оформленные так же, как в предыдущем случае; отличие в том, что текст, печатающийся в колонтитулах (в стандартном случае это номера и названия разделов документа), не порождается Л^AT_EXом автоматически, а задается пользователем в явном виде.

Если основной стиль — `article`, то по умолчанию страницы оформляются стилем `plain`, в двух других основных стилях — стилем `headings`. «Стиль» `myheadings` мы рассмотрим в разделе 6 главы IX, посвященной модификации стандартных стилей.

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...)

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, переделав оформление колонтитулов или номеров страниц совершенно радикально. Речь об этом пойдет в главе IX.

2. Поля, размер страницы и прочее

Стандартные ЛАТЭХовские стили самостоятельно устанавливают значения таких параметров, как ширина и высота страницы, размеры полей и пр. Если эти значения Вас не устраивают, их можно изменить. В настоящем разделе рассказано, как это сделать.

Размеры текста на странице, полей и прочее задаются различными параметрами со значением длины (см. раздел 1.2.6 по поводу ТЭХовских параметров). Прежде, чем мы скажем, какими именно, надо предупредить читателя, что значения параметров, о которых ниже пойдет речь, можно менять только в преамбуле документа (или в стилевом файле, если Вы создали собственный стилевой файл). Изменение этих параметров после `\begin{document}` в одних случаях вообще ничего не даст, а в других — приведет к нелепым результатам.

2.1. 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 Вам нужно, чтобы ширина текста на странице равнялась 7 см, то напишите в преамбуле так:

```
\textwidth=7cm
```

При изменении ширины текста часто приходится менять и поля. В ЛАТЭХе предусмотрен параметр, регулирующий размер левого поля (коль скоро левое поле и `\textwidth` заданы, правое поле, как Вы догадываетесь, определяется автоматически). Способ задания левого поля зависит от того, является ли набор в данном стиле «двусторонним» или нет. На стр. 98 объяснялось, что при двустороннем наборе на страницах с четными и нечетными номерами оставляются разные поля. В стандартном стиле `book` набор является двусторонним всегда, а в двух других стандартных стилях набор по умолчанию односторонний, но он будет двусторонним, если указать стилевую опцию `twoside`.

При одностороннем наборе величина левого поля задается параметром `\oddsidemargin`. При этом левое 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 для страниц с

четными номерами, то размеры левого поля для них задаются параметром `\evensidemargin` (что не должно удивлять читателей, знающих английский).

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`).

2.2. 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 (если Вы доберетесь до главы IX, то узнаете, что размер этого места задается параметром `\headheight`), плюс пустое пространство, равное отступу между колонтитулом и основным текстом (оно обозначается `\headsep`).

Высота текста задается параметром `\textheight`. При исчислении этого размера не учитываются ни номера страниц, ни колонтитулы, так что, если они предусмотрены стилем, полная высота текста на странице будет больше, чем `\textheight`.

Высоту страницы также можно изменять, присваивая в преамбуле параметру `\textheight` новое значение, но если стиль предусматривает, что все страницы должны иметь одинаковую высоту (см. стр. 98 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 строки¹ 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 книги [2]). Так или иначе, значение `\textheight` следует устанавливать таким образом, чтобы отношение

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

было целым числом. Значения `\baselineskip` и `\topskip`, устанавливаемые в различных стилях, приведены в приложении Г.

2.3. Сдвиг страницы как целого

Практически при печати Вы можете иногда с удивлением обнаружить, что реальные расстояния от текста до края листа не такие, как предписано параметрами наподобие `\topmargin`. Дело в том, что у принтера, которым Вы пользуетесь, могут быть свои, не всегда согласующиеся с реальностью, представления о том, где находится край листа бумаги. Чтоб

¹Точнее говоря, от ее базисной линии: см. главу VIII

эти представления соответствовали реальности, понадобится настройка принтера и/или dvi-драйвера, используемого Вами для распечатки. Если Вам жаль тратить время на такую работу, можно просто изменить расположение всей страницы в целом на печатном листе. Для этого следует установить (в преамбуле) значения двух Т_EXовских 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.

3. Разделы документа

Работая с Л^AT_EXом, разумно делать заголовки и нумерацию разделов не вручную, а с помощью специальных команд. Сначала разберем, как ими пользоваться, на примере команды `\section`.

3.1. 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 стилем, и в оглавление, если Вы дадите команду «создать оглавление» — см. ниже стр. 106). Промежутки между разделами, их нумерация, те же колонтитулы — все это делается автоматически.

Кроме обязательного аргумента, у команды `\section` предусмотрен и необязательный. Необязательный аргумент идет перед обязательным; в нем записывается вариант заголовка, предназначенный для оглавления и колонтитулов (если стиль предусматривает, что заголовок войдет в колонтитул). Обычно этот вариант — просто сокращенный заголовок. Пример:

```
\section[0 слонах]{Кое-что о слонах}
```

Необходимость в сокращенном варианте заголовка возникает, когда оказывается, что заголовков 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. стр. 17). 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 разделе 3.2) — большие и добрые животные.

```
\section{Кое-что о слонах}
\label{elephants}
```

В этом разделе нашей книги речь пойдет в основном о слонах.

Слоны (см. \ определение в разделе~\ref{elephants} --- большие и добрые животные.

Как обычно с командами автоматической генерации ссылок, при первом запуске \LaTeX а будет выдано сообщение о том, что метка неизвестна, а в дальнейшем, если номер помеченного раздела изменится, \LaTeX выдаст предупреждение о том, что надо запустить его еще раз.

Иногда хочется, чтобы сокращенный вариант заголовка попал только в колонтитул, а в оглавлении был его полный вариант. Как этого добиться, рассказано в разделе 6 главы IX (см. особенно стр. 202 и далее).

У команды `\section` есть вариант «со звездочкой» (см. стр. 16). 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.

В разделах, создаваемых вышеописанными командами, первый абзац набирается без абзацного отступа (за исключением самого мелкого раздела `\subparagraph`), причем \LaTeX устроен таким образом, что создать этот отступ Вам так просто не удастся. Если Вы хотите, чтобы отступ в первом абзаце все-таки присутствовал, обратитесь к главе IX, посвященной модификации стандартных стилей.

3.2. Какие бывают разделы документа

Теперь перечислим все команды для задания разделов документа, предоставляемые стандартными стилями \LaTeX а. Большинство из них работают 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 в виду, что команда `\chapter` («глава») в стиле `article ne` определена (благодаря этому обстоятельству статью легко переделать в главу книги), остальные команды определены во всех трех стандартных стилях.

Стандартные стили обеспечивают нумерацию разделов, при которой более мелкий раздел «подчинен» более крупному: когда, например, начинается новая `\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 стилей 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 стилям, 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.

3.3. Изменение стандартных заголовков

Возможно, Вы уже обратили внимание, что главы, создаваемые ЛАТЭХом с помощью команды `\chapter`, так и называются «Chapter», а не «Глава». Это — один из нескольких случаев, когда стили ЛАТЭХа используют для различных стандартных надписей английские слова. Если вместе с ЛАТЭХом Вы получили русифицирующий стилевой файл, то у Вас есть возможность указать стилевую опцию, делающую эти надписи русскими. Если такой русификации у Вас нет, то придется решить эту проблему своими силами. Для этого в преамбулу документа надо внести следующую команду:

```
\renewcommand{\chaptername}{Глава}\chaptername
```

(делать это надо только в том случае, если команда `\chapter` предусмотрена в используемом Вами стиле). Команда `\renewcommand`, используемая для переопределения значений уже существующих команд, объясняется в главе VII.

Приведем заодно список остальных надписей, делаемых ЛАТЭХом по-английски, вместе с их русскими переводами и командами, которые надо переопределить с помощью `\renewcommand` для того, чтоб на печати появлялись именно эти переводы. В нескольких ближайших разделах мы объясним, как именно получить на печати упоминаемые в этой таблице список таблиц, список иллюстраций и т. п.

<code>\contentsname</code>	Contents	Оглавление
<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	Список литературы
<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

3.4. Аннотация, приложение

В стилях `article` и `report` предусмотрена возможность оформить аннотацию ко всему документу. Это делается с помощью окружения `abstract`. До начала основного текста следует поместить текст аннотации, заключенный между `\begin{abstract}` и `\end{abstract}`. Этот текст будет автоматически озаглавлен «Abstract», если не менять стиля. Чтобы получить другой заголовок, переопределите команду `\abstractname` (см. предыдущий пункт).

Команда `\appendix` означает, что с этого места начинается приложение к документу. Сама она никакого текста не производит, а делает вот что:

- Начинает заново нумерацию разделов документа;

- «Самые крупные» разделы документа (то есть `\section` в стиле `article` и `\chapter` в двух других стандартных стилях) начинают нумероваться не цифрами, а прописными латинскими буквами.
- Если определена команда `\chapter`, то главы будут называться не «Глава», а так, как определено в команде `\appendixname` (см. предыдущий пункт).

4. Титульный лист, оглавление, список литературы

4.1. Титул

Для того, чтобы оформить заголовок ко всему документу, надо сделать две вещи: задать информацию для заголовка (автор, название и т. п.) и дать ЛАТЭХу команду этот заголовок сгенерировать. Второе делается с помощью команды `\maketitle`. Она создаст титульный лист, если это предусмотрено стилем (стало быть, для стилей `report` и `book` — всегда, для стиля `article` — если указана стилевая опция `titlepage`). Если титульный лист стилем не предусмотрен, то команда `\maketitle` разместит заданную Вами информацию об авторе, заглавии и прочем на первой странице (выбрав подходящие шрифты и сделав подходящие интервалы между титульной информацией и текстом).

Так как команда `\maketitle` генерирует текст, ее нельзя помещать в преамбуле документа.

Теперь объясним, как задавать ЛАТЭХу 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 `\\` (кстати, в этой ситуации защищать ее с помощью `\protect` не требуется — см. стр. 111); если этого не сделать, заглавие будет разбито на центрированные строки автоматически, как если бы это был абзац в окружении `center` (см. стр. 85, а также пример на стр. 16).

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 стилем) 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 ЛАТЭХ напечатает на титуле дату своего запуска, причем по-английски.

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 сделать сноски. 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 в тексте, он должен состоять из одного абзаца).

Сноски будут напечатаны внизу титульного листа (или первой страницы, если титульный лист не предусмотрен). Пример задания сносок:

```
\author{Борис Заходер}  
\title{Винни-Пух\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 \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. \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.

4.2. Оглавление

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

Стало быть, оглавление, генерируемое \LaTeX ом, всякий раз будет «на шаг отставать» от реального положения дел. Чтобы учесть все возможные изменения и получить верное оглавление, надо будет в самом конце работы над текстом запустить \LaTeX еще раз (напоминания об этом \LaTeX не выдаст).

Все оглавление в целом будет озаглавлено словом, определяемым командой `\contentsname` (см. раздел 3.3).

Если Вас не устраивает стандартный стиль оформления оглавления, прочтите в последней главе, как его можно изменить.

4.3. Список литературы

\LaTeX предоставляет 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 (будем для краткости называть их «источниками»).

Список литературы оформляется как окружение `thebibliography`. Это окружение имеет обязательный аргумент — номер элемента библиографии, который займет больше всего

места на печати (в стандартных шрифтах все цифры имеют одинаковую ширину, так что достаточно привести в качестве аргумента, например, номер 99, если в списке литературы будет заведомо меньше 100 источников).

Каждый элемент списка литературы вводится командой `\bibitem`. У нее есть один обязательный аргумент — Ваше условное обозначение. В качестве такого обозначения можно использовать любую последовательность из букв и цифр.

В тексте ссылка на элемент списка литературы 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 через запятую обозначения для тех источников, на которые Вы хотите сослаться. Приведем пример (в котором для экономии места мы опустили заголовок «Список литературы»):

В книге [3, Глава 1] описана встреча Винни-Пуха с несколькими пчелами. В [2, 1] приведены другие сведения о медведях.

[1] М.Е.Салтыков-Щедрин.
Медведь на воеводстве.

[2] Л.Н.Толстой. Три медведя.

[3] А.А.Милн. Винни-Пух.

```

В книге~\cite[Глава~1]{Winnie}
описана встреча Винни-Пуха
с несколькими пчелами.
В~\cite{med3,voevoda}
приведены другие
сведения о медведях.
\begin{thebibliography}{99}
\bibitem{voevoda} М.Е.Салтыков-Щедрин.
Медведь на воеводстве.
\bibitem{med3} Л.Н.Толстой.
Три медведя.
\bibitem{Winnie}
А.А.Милн. Винни-Пух.
\end{thebibliography}

```

В этом примере Вы также можете видеть команду `\cite` с необязательным аргументом: он ставится перед обязательным; в квадратных скобках записывается текст, который будет через запятую напечатан после номеров ссылок.

Как это обычно и происходит с автоматически генерируемыми ЛАТЭХом ссылками, после первого запуска программы Вы увидите сообщение о том, что ссылки не определены. Если в дальнейшем в процессе работе над текстом нумерация ссылок изменится, ЛАТЭХ 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}{SGA7}
\bibitem[EGA]{Groth} A.Grothendieck,
J.Dieudonn'e. \ 'El'ements de
G\ 'eom\ 'trie Alg\ 'ebrique.
\bibitem[SGA7]{Deligne}
P.Deligne, N.Katz. Groupes de
monodromie en G\ 'eom\ 'etrie
Alg\ 'ebrique.

```

```
\end{thebibliography}
```

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

Списку литературы в целом \LaTeX автоматически дает заглавие, определяемое командой `\refname` в стиле `article` и `\bibname` в стилях `report` и `book` (см. раздел 3.3). Если Вас не устраивает, что это название — английское, его можно переопределить (см. там же).

4.4. Предметный указатель

\LaTeX окажет Вам помощь в создании предметного указателя к Вашему документу. В отличие, однако, от списка литературы, который при использовании вышеописанных команд `\cite` и `\bibitem` получается совершенно автоматически, процесс создания указателя автоматизирован в \LaTeX е не полностью. Именно, Вы можете сделать две вещи:

- Если Вам уже известно, на каких страницах расположены те термины, на которые Вы собираетесь сослаться в указателе, Вы можете организовать печать предметного указателя с помощью окружения `theindex`. Если предметный указатель должен завершать документ, то можно, на худой конец, напечатать весь документ, кроме указателя, и вручную выписать требуемые номера страниц².
- Если первый способ Вас не устраивает, то можно специальным образом пометить в файле термины, на которые Вы собираетесь сослаться в предметном указателе. При этом средствами \LaTeX а создается полуфабрикат, из которого предметный указатель получится после некоторой дополнительной обработки.

Разберем эти две возможности последовательно.

Внутри окружения `theindex` каждый элемент указателя вводится командой `\item`; команды `\subitem` и `\subsubitem` вводят элементы указателя, печатающиеся с дополнительными отступами (обычно это — уточнения к заглавному слову). Наконец, команда `\indexspace` создает дополнительный вертикальный пробел (его можно использовать для отделения различных разделов указателя друг от друга):

Компьютеры 25 и далее	<code>\begin{theindex}</code>
ИВМ-совместимые 28	<code>\item Компьютеры 25 и далее</code>
ремонт 35	<code>\subitem ИВМ-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>
	<code>\indexspace</code>
Кошки 120	<code>\item Кошки 120</code>
Собаки 140–156	<code>\item Собаки 140--156</code>
	<code>\end{theindex}</code>

²У Вас может возникнуть искушение пометить все места, на которые будете сослаться, с помощью команды `\label`, а в окружении `theindex` сослаться на них с помощью `\pageref`. При этом, однако, есть опасность, что \TeX у не хватит памяти для обработки такого огромного числа ссылок.

Предметный указатель, получаемый из окружения `theindex`, печатается \LaTeX ом в две колонки (даже тогда, когда сам документ печатается в одну колонку). Кроме того, \LaTeX автоматически дает списку литературы заглавие, определяемое командой `\indexname` (см. раздел 3.3). Если Вас не устраивает, что это название — английское, его можно переопределить (см. там же).

Теперь рассмотрим вторую возможность — как с помощью \LaTeX а автоматически создать полуфабрикат предметного указателя. Для этого 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`. У этой команды один обязательный аргумент — текст Вашей пометки (как правило, такая пометка — это просто заглавное слово будущего предметного указателя). \LaTeX запишет информацию о том, на какие страницы попали Ваши пометки, 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}
```

Полученный таким образом `idx`-файл — это и есть полуфабрикат списка литературы, созданный \LaTeX ом. Использовать этот полуфабрикат, однако же, еще нельзя по следующим причинам:

- Ссылки в `idx`-файле расположены не по алфавиту (или каким-то другим разумным образом), а записаны «в порядке поступления».
- В `idx`-файле может присутствовать несколько строк с одним заглавным словом и ссылками на разные страницы; в готовом указателе естественно было бы такие ссылки объединить.
- Все строки `idx`-файла равноправны, в нем отсутствует иерархия ссылок вроде той, что получается при использовании команд наподобие `\subitem` в окружении `theindex`.
- Команда `\indexentry`, с которой начинается каждая строка `idx`-файла, *не* определена в \LaTeX е (это сделано сознательно!).

Поэтому, получив `idx`-файл, Вы должны его каким-либо образом обработать; идеально, если в результате получится файл с отсортированными (по алфавиту, скажем) терминами и с командами `\item`, `\subitem` и т. д., так что можно будет написать

```
\begin{theindex}
\input <имя_файла>
\end{theindex}
```

Если Вы немного умеете программировать, Вы сможете осуществить такую обработку программным путем самостоятельно; кроме того, к реализациям Т_ЭХа нередко прилагаются программы для обработки `idx`-файлов (учтите, однако, что, если такая программа получена с Запада, то слова с русскими буквами она, наверное, будет сортировать неправильно). Чтобы установить иерархию терминов (`\subitem` и т. п.), Вам придется поработать вручную (как отмечал Дональд Кнут, составление предметного указателя — процесс творческий, и полностью передоверять его компьютеру не следует).

Как быть, если программы для обработки `idx`-файлов у Вас нет и самостоятельно написать ее Вы не можете? Во-первых, отсортировать строки `idx`-файла можно и без специальной программы, средствами текстового редактора (в хороших редакторах такая возможность обычно предусмотрена). После этого остается проблема, что делать с командами `\indexentry`. Если Ваших программистских умений не хватает на то, чтобы преобразовать каждое `\indexentry` в `\item`, то есть еще одна возможность: осуществить это преобразование *средствами самого ЭТ_ЭХа*. Именно, после того, как Вы отсортируете строки `idx`-файла (и сохраните, для надежности, отсортированный файл под другим именем, скажем, `myindex.tex`), надо *определить* оставленную неопределенной команду `\indexentry` таким образом, чтобы она делала ту же работу, которую призван делать `\item`. Для этого надо написать в преамбуле следующее:

```
\newcommand{\indexentry}[2]{\item #1 #2}
```

После этого Т_ЭХ 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 главы VII, в которой подробно рассмотрен процесс определения новых команд, Вы поймете, почему все получается именно так.

В аргументе команды `\index` может быть любой текст и любые Т_ЭХовские команды — все это в неизменном виде будет переписано в `idx`-файл. Единственное ограничение — не должно быть «несбалансированных» фигурных скобок.

Если в файле присутствуют команды `\index`, но в преамбуле нет команды `\makeindex`, то в `idx`-файл ничего записываться не будет.

Наконец, еще одна тонкость: команду `\index` нельзя использовать внутри необязательного аргумента таких команд, как `\caption` или `\section`, `\chapter` и т. п.

4.5. Перемещаемые аргументы и хрупкие команды

Если в аргументе команды `\section` (или любой другой ЛАТЭХовской команды для создания раздела) присутствует не только текст, но и ТЭХовские 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.

Пусть Вы хотите включить в заголовок раздела ссылку на какую-то страницу документа. Для этого Вы, как обычно, помечаете то место, на которое хотите сослаться, с помощью команды `\label`, а в заголовок раздела включаете команду `\pageref` со ссылкой на помеченное место. Для того, однако же, чтоб ЛАТЭХ обработал Ваш текст правильно, надо в аргументе команды `\section` написать не просто `\pageref`, но `\protect\pageref`. В итоге Ваш исходный файл должен будет выглядеть примерно так (несущественные для нас части файла мы заменили точками):

```
\documentstyle{article}
\begin{document}
\tableofcontents

Это --- начало документа.
.....
Здесь написано что-то очень
важное.\label{metka}
.....
\section{Возвращаясь к напечатанному
        на стр.\ \protect\pageref{metka}}
.....
\end{document}
```

Если убрать в этом файле `\protect`, то после двух запусков ЛАТЭХа для его обработки на экране будет появляться появится загадочное сообщение об ошибке.

Такого рода ситуация может возникать, когда ТЭХовская команда является частью текста, который будет записан в специальный файл и использован при следующем запуске ЛАТЭХа (в нашем случае заголовок раздела записывается в файл с расширением `toc` для последующего использования в оглавлении). Если аргументом команды (в нашем случае команды `\section`) является такой текст, то этот аргумент называется *перемещаемым аргументом*; команды, которые, будучи использованы внутри перемещаемого аргумента, могут вызвать ошибку, называются *хрупкими командами*.

Из тех ЛАТЭХовских команд, которые могут реально понадобиться внутри заголовка раздела, большинство хрупкими не являются; наряду с `\pageref` и `\ref`, хрупкой является также команда `\`, которая может понадобиться для указания места разрыва строки в заголовке. Если Вы сомневаетесь, хрупкая или нет какая-то конкретная команда, можете спокойно ставить перед ней `\protect` — плохого от этого не произойдет.

Нижеследующие команды *не* являются хрупкими и не нуждаются в защите с помощью команды `\protect`:

- Команды для диакритических знаков (см. стр. 63);
- Команды для смены текущего шрифта;
- Команды для установки промежутков вручную (см. стр. 62).

Рис. IV.1. К. Малевич. Белый квадрат на белом фоне

5. Плавающие иллюстрации и таблицы

Если Вы хотите разместить в своем тексте иллюстрацию (в простейшем случае — место для приклеивания картинки, но это может быть и «псевдорисунок», создаваемый с помощью ЛАТЭХовского окружения `picture`, и импортированный графический файл, если Ваш `dvi`-драйвер предоставляет такую возможность), то можно создать для нее «пустое место, не пропадающее при разрыве страницы», с помощью команды `\vspace*` (см. раздел III.7.3). Если, однако, таких команд будет много, то будет трудно (или вообще невозможно) найти подходящие места для разрыва страниц. В этом случае удобнее воспользоваться окружением `figure`. Стоящий между `\begin{figure}` и `\end{figure}` текст автоматически размещается ЛАТЭХом в таком месте, где он укладывается целиком (не переходя со страницы на страницу); это может быть не на «своей» странице, а позже. В последнем случае говорят, что иллюстрация «всплыла» на следующей странице; именно поэтому окружение `figure` называют еще «плавающая иллюстрация».

Команда `\caption` позволяет сделать подрисовочную подпись. Эта команда имеет один обязательный аргумент — текст подписи. На печати подпись состоит из слова, определенного командой `\figurename` («Fig.», если не переопределять эту команду — см. раздел 3.3), порядкового номера иллюстрации, присвоенного ей ЛАТЭХом, 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 ей подпись. Разумно поэтому ставить команду `\caption` либо в конце окружения `figure` (тогда подпись будет размещена под иллюстрацией), либо в его начале (подпись появится над иллюстрацией).

Если команду `\label` поместить внутри окружения `figure`, то команда `\ref` будет генерировать номер иллюстрации. Например, рисунок IV.1 на стр. 112 получился так:

```
Рис.~\ref{void} --- это шедевр супрематизма.
\begin{figure}
\vspace{4cm}
\caption{К.~Малевич. Белый
квадрат на белом фоне}\label{void}
\end{figure}
```

У окружения `figure` предусмотрен необязательный аргумент, с помощью которого можно высказать ЛАТЭХу 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 обязательного аргумента, это равносильно записи

```
\begin{figure}[tbp]
```

Если Вы будете злоупотреблять внешне привлекательным необязательным аргументом `h` у окружения `figure`, то возникнут те же затруднения с поиском хороших мест разрывов страниц, как при переиспользовании команд вроде `\vspace*`.

При наборе текста в две колонки полезно использовать не только само окружение `figure`, но и его вариант «со звездочкой» (см. раздел I.2.9): 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 одну колонку (без звездочки получилось бы шириной в целую страницу). Если окружение открывается командой `\begin{figure*}`, то и закрываться оно должно командой со звездочкой.

Окружение `table` определяет «плавающие таблицы». Все свойства этого окружения дословно совпадают с соответствующими свойствами окружения `figure`, за одним-единственным исключением: подпись, генерируемая командой `\caption`, начинается со слова, определенного в команде `\tablename` (см. раздел 3.3), и переопределять, при необходимости, надо именно эту команду.

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. стр. 106): материал для этих списков собирается в специальные файлы с расширениями `.lof` (для иллюстраций) и `.lot` (для таблиц); при каждом запуске \LaTeX информация, записанная в этих таблицах, относится к предыдущему запуску, так что в самом конце может понадобиться запустить \LaTeX лишний раз; наконец, команда `\caption` может принимать необязательный аргумент — вариант подписи под иллюстрацией или таблицей, предназначенный для включения в список иллюстраций или таблиц соответственно. Этот необязательный аргумент записывается (в квадратных скобках, как обычно) *перед* обязательным.

Как окружение `figure` не рисует картинок, так и окружение `table` только размещает таблицу на страницах документа, но не создает ее текста. Как набирать таблицы в \LaTeX е, мы расскажем в отдельной главе.

6. Заметки на полях

\LaTeX дает возможность делать заметки на полях страницы. Для этого используется команда `\marginpar` с единственным обязательным аргументом — текстом заметки. Предыдущий текст выглядел в исходном файле так:

Например
вот
это —
замет-
ка на
полях.

...делать заметки на полях. Для этого используется `\marginpar`{Например, вот это --- заметка на полях.} команда...

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 (то есть, основной стиль — `article` или `report` и нет стилевой опции `twoside`), то заметки выводятся по умолчанию на правое поле. Если документ печатается в одну колонку, но в «двустороннем» стиле — то на внешнее поле (правое, если страница правая на развороте, и левое в противном случае). Если документ печатается в две колонки, то заметка всегда выводится на ближайшее поле (ближайшее к той колонке, в которую попала заметка).

У команды `\marginpar` предусмотрен и необязательный аргумент. Он размещается *перед* обязательным; если эта команда использована с необязательным аргументом, то текст, выводимый 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. стр. 32 по поводу команд, генерирующих стрелки в математических формулах).

ИТЭХ старается поместить заметки на полях на том же уровне, что и текст, к которым они относятся, но, если этих заметок на каждой странице получается помногу (как, например, в поэмах С. Т. Кольриджа «Сказание о старом мореходе» или В. В. Маяковского «Про это»), то некоторые из них, во избежание наложений, будут сдвинуты вниз, а иногда даже перенесены на другую страницу (если это прискорбное событие произойдет, ИТЭХ выдаст предупреждение 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 в заметках на полях.
\marginparsep   : Расстояние между полем и заметками.
\marginparpush  : Минимальное расстояние по вертикали между соседними заметками.
```

Значения этих параметров устанавливаются автоматически, в зависимости от используемого стиля. Вам может понадобиться их изменить, если Вы меняете размер полей и/или ширину текста и при этом хотите пользоваться командой `\marginpar`.

В некоторых ситуациях команду `\marginpar` применять нельзя. Например, она не может появиться внутри аргумента команды `\mbox` или внутри окружения, предназначенного для верстки таблиц (см. главу VI). Если Вы все-таки попытаетесь сделать заметку на полях к такому «запрещенному» месту, то ИТЭХ выдаст сообщение об ошибке.

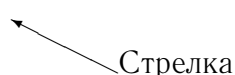
Глава V.

Псевдорисунки

Когда создавался Т_ЕX, а начиналось это в 1978 году, компьютерная графика была развита еще слабо. Поэтому операция по включению в текст рисунков в виде графических файлов в Т_ЕXе не стандартизирована. Точнее говоря, Т_ЕX допускает импорт графического файла в текст с помощью команды `\special`, в аргументе которой содержится информация об импортируемом файле, но способ задания этой информации не стандартизирован и зависит от конкретной реализации Т_ЕXа (именно, от используемых dvi-драйверов), что снижает переносимость Т_ЕXовских файлов (исходный текст, в котором `\special` не используется, обрабатывается совершенно одинаково на всех реализациях Т_ЕXа на любом компьютере). Чтобы как-то сгладить это неудобство, создатель Л^AT_ЕXа Лесли Лэмпорт предусмотрел возможность создания примитивных рисунков, состоящих из прямых и наклонных (с ограниченным репертуаром наклонов) линий, стрелок и окружностей. В этой главе мы расскажем, как создавать такие «псевдорисунки».

1. Создание псевдорисунка и размещение в нем объектов

Псевдорисунки создаются с помощью окружения `picture`. Изучение этого окружения удобно начать с примера.

	<pre>\begin{picture}(110,50) \put(55,35){\vector(-2,1){40}} \put(55,35){Стрелка} \end{picture}</pre>
---	--

Разберем исходный текст, создавший этот «рисунок»: стрелку с надписью. На каждый псевдорисунок Л^AT_ЕX должен отвести в тексте определенное место (после чего сам рисунок вполне может и выйти за пределы отведенного места: все зависит от того, что и где Вы будете «рисовать»). Эти размеры задаются в *круглых* скобках через запятую немедленно после `\begin{picture}`, сначала ширина, затем высота (команды, связанные с псевдорисунками — единственные в Л^AT_ЕXе, у которых в определенных случаях аргумент ставится *не* в фигурных скобках). Между скобками, запятой и числами, задающими размеры псевдорисунка, не должно быть пробелов (помните, что конец строки также воспринимается Т_ЕXом как пробел; если переноса на другую строку не избежать, воспользуйтесь знаком `%` для устранения получающегося пробела, как в примере на стр. 11). По умолчанию длина и ширина псевдорисунка, и вообще все относящиеся к псевдорисункам размеры, задаются в пунктах

(так и сделано в нашем примере). Можно указать любую единицу измерения размеров, относящихся к псевдорисункам: для этого надо изменить значение параметра `\unitlength` (см. стр. 14 и далее по поводу параметров, являющихся длинами): если мы хотим, чтобы длины измерялись в миллиметрах, надо написать в преамбуле

```
\unitlength=1mm
```

(но не просто `mm`!). Размеры могут быть не только целыми, но и дробными числами, в которых нужно использовать десятичную точку (но не запятую!).

Итак, место на псевдорисунке выделено. Чтобы поместить что-то на этот псевдорисунк, 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 второй из наших команд `\put` этот объект был просто текстом, и соответственно в фигурных скобках только этот текст и был; для первой из команд, размещавшей на рисунке стрелку, в фигурных скобках помещается нечто более сложное: описание этой стрелки. В следующем разделе мы разберем, как такие описания устроены.

Когда мы говорили о координатах объекта, имелись в виду координаты так называемой «точки отсчета» на этом объекте. Если объект — текст, то точка отсчета — его левый нижний угол. Иногда при размещении текста удобней задать координаты его правого, а не левого нижнего угла. Чтобы так сделать, можно воспользоваться командой `\llap` с одним аргументом — текстом, чья точка отсчета будет в правом нижнем углу. Пример:

Кошка Кошка

```
\begin{picture}(110,40)
\put(52,20){\bf Кошка}
\put(50,20){\llap{\sf Кошка}}
\end{picture}
```

Точка отсчета стрелки — ее начало. Когда пойдет речь о других объектах, размещаемых на псевдорисунке, мы будем указывать, где расположены их точки отсчета.

Еще немного об общих правилах, относящихся к окружению `picture`. Во-первых, внутри этого окружения не должно быть пустых строк. Во-вторых, необходимо сказать о том, как окружение `picture` взаимодействует с окружающим текстом. Весь псевдорисунк, порождаемый этим окружением, рассматривается Т_ЕXом как одна большая буква, ширина и высота которой заданы в скобках через запятую после `\begin{picture}`, так что если окружение `picture` встретилось в середине абзаца, эта «буква» будет помещена в строку, причем соседние строки раздвинутся, чтобы она поместилась. Если это не то, чего Вы хотите, — найдите окружение `picture` между абзацами (после пустой строки или команды `\par`). Можно также поместить окружение `picture` внутри окружения наподобие `flushright` или `center` — при этом Т_ЕX автоматически установит разумные интервалы между псевдорисунком и окружающим текстом. Совершенно безбоязненно можно помещать окружение `picture` внутри «плавающего» окружения `figure` или `table` (см. раздел IV.5).

Кроме текста, на псевдорисунках можно размещать отрезки, стрелки, окружности, круги и овалы (прямоугольники с закругленными углами). Далее мы опишем, как задавать эти объекты.

2. Отрезки и стрелки

Отрезки задаются с помощью команды `\line`. ЛАТ_EXу надо сообщить, каков наклон отрезка и каков его размер. Вот пример команды `\put`, выводящей отрезок:



```
\begin{picture}(100,50)
\put(60,50){\line(1,-2){20}}
\end{picture}
```

Как мы уже понимаем, здесь создается псевдорисунок размером 100×50 пунктов, на который наносится отрезок с концом в точке с координатами $(60, 50)$. Наклон отрезка задается парой целых чисел, расположенных в *круглых* скобках через запятую непосредственно после `\line`. Отношение этих чисел должно быть равно «угловому коэффициенту» отрезка (тангенсу угла наклона к горизонтали); в нашем случае эти числа суть $(1, -2)$, это означает, что отрезок отклоняется «на одну единицу вправо и на две единицы вниз». Если эти числа $(1, 0)$, то отрезок горизонтален, если $(0, 1)$, то отрезок вертикален.

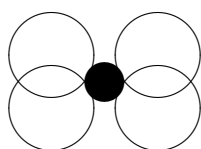
Размер отрезка задается в фигурных скобках после круглых скобок, в которых задан наклон. Этот размер, вообще говоря, — *не* его длина, но длина его проекции на горизонтальную ось (кроме случаев, когда отрезок вертикален — тогда задается его длина по вертикали).

Длину отрезка можно (если она на слишком мала) задавать произвольно, а вот наклон — нет. Два целых числа, задающие наклон, не должны превосходить 6 по абсолютной величине, и, кроме того, не должны иметь общих делителей, кроме 1 (это последнее условие репертуар возможных наклонов не ограничивает).

Стрелки задаются с помощью команды `\vector`, которая нам уже встречалась в примере. Синтаксис этой команды совершенно такой же, как у команды `\line`: в круглых скобках пишется пара чисел, задающая наклон стрелки, а затем в фигурных скобках параметр, задающий ее размер (длина проекции на горизонтальную ось, если стрелка не вертикальна, и длина проекции на вертикальную ось, если стрелка вертикальна). Отличие от команды `\line` в том, что репертуар возможных наклонов стрелок еще более ограничен, чем у отрезков: целые числа, задающие наклон, не должны превосходить 4 по абсолютной величине (и по-прежнему не должны иметь общих делителей). Точкой отсчета стрелки является ее начало.

3. Окружности, круги и овалы

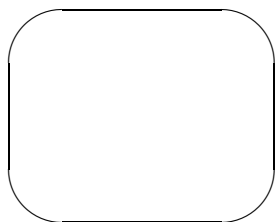
Окружность задается командой `\circle`, а круг (сплошной черный кружок) — ее вариантом «со звездочкой» `\circle*`. У этих команд единственный аргумент — диаметр круга или окружности. Как обычно, он задается в единицах, равных значению параметра `\unitlength` (по умолчанию — в пунктах). Точкой отсчета окружности или круга является центр. Вот пример картинки с окружностями и кругами:



```
\begin{picture}(100,80)
\put(30,30){\circle{30}}
\put(70,30){\circle{30}}
\put(30,50){\circle{30}}
\put(70,50){\circle{30}}
\put(50,40){\circle*{20}}
\end{picture}
```

Количество реально возможных диаметров кругов ограничено. Если окружности или круга с диаметром, указанным в качестве аргумента команды `\circle` или `\circle*`, в \TeX 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 — длина и ширина овала. Эти аргументы задаются в *круглых* скобках через запятую. Точка отсчета овала — его центр. Пример:

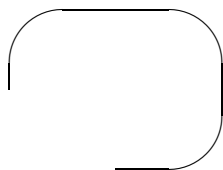


```
\begin{picture}(100,80)
\put(50,40){\oval(100,80)}
\end{picture}
```

Кроме того, возможны и «неполные» овалы, представляющие собой половинки или четвертинки от полных. Чтобы задать такой неполный овал, надо задать команде `\oval` необязательный аргумент (в квадратных скобках, после обязательного). Для задания половинки овала этот аргумент должен быть одной из следующих букв:

- t Верхняя половина
- b Нижняя половина
- r Правая половина
- l Левая половина

Для задания четвертинки овала необязательный аргумент команды `\oval` должен быть сочетанием двух из этих букв (например, `tr` для верхней правой четвертинки). Точка отсчета усеченного овала расположена там же, где точка отсчета соответствующего ему полного овала. Вот пример картинки с усеченными овалами:



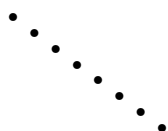
```
\begin{picture}(100,80)
\put(50,40){\oval(80,60)[t]}
\put(50,40){\oval(80,60)[br]}
\end{picture}
```

4. Дополнительные возможности

Иногда бывает нужно нанести на псевдорисунок несколько регулярно расположенных объектов. В этом случае, вместо того, чтобы много раз писать `\put`, удобно воспользоваться командой `\multiput`. Она располагает на псевдорисунке несколько одинаковых объектов на равных расстояниях. Синтаксис этой команды таков:

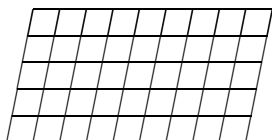
$$\text{\multiput}(x, y) (\Delta x, \Delta y) \{n\} \{\text{объект}\}$$

Здесь x и y — координаты первого из размещаемых объектов (как и в обычной команде `\put`), Δx и Δy — насколько каждый следующий объект будет сдвинут относительно предыдущего по горизонтали и вертикали, n — количество объектов, которые надо разместить, и, наконец, *объект* — это, как и у команды `\put`, описание того, что мы размещаем на рисунке. Пример:



```
\begin{picture}(100,80)
\multiput(10,70)(8,-6){8}%
{\circle*{3}}
\end{picture}
```

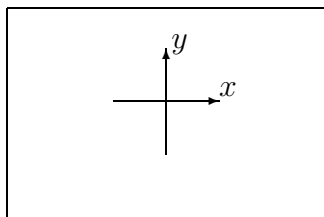
Обратите внимание на использование знака процента для удаления нежелательного пробела, создаваемого концом строки. Вот еще один пример; здесь с помощью команды `\multiput` рисуется решеточка:



```
\begin{picture}(100,50)
\multiput(0,0)(10,0){10}%
{\line(1,5){10}}
\multiput(0,0)(2,10){6}%
{\line(1,0){90}}
\end{picture}
```

С формальной точки зрения оба вышеприведенных примера совершенно правильны. Практически, однако, такое использование команды `\multiput` ведет к неоправданным затратам машинного времени. Например, каждый из наклонных отрезков во втором примере собирается из маленьких символов, причем $\text{T}_\text{E}_\text{X}$ приходится повторять эту скучную операцию 10 раз. Разумнее было бы собрать этот отрезок лишь единожды, а дальше его просто копировать. $\text{L}_\text{A}_\text{T}_\text{E}_\text{X}$ позволяет это сделать с помощью конструкции «блоковых переменных». Мы расскажем об этом в главе VIII.

Иногда, когда псевдорисунок достаточно сложен, удобно применить следующий прием: задать в качестве аргумента одной из команд `\put` целое окружение `picture` (точкой отсчета будет служить левый нижний угол). При этом Вы сможете отсчитывать координаты объектов на этом «подрисунке» относительно самого подрисунка, а не внешнего рисунка, что часто бывает проще; кроме того, если Вам понадобится сдвинуть этот «подрисунок» как единое целое, то для этого будет достаточно изменить аргументы в одной-единственной команде `\put`. Вот пример рисунка с подрисунком (будем считать, что это классная доска, на которой нарисованы оси координат):



Этому рисунку соответствовал такой исходный текст:


```

\begin{picture}(120,80)
% Края доски:
\put(0,0){\line(1,0){120}}
\put(0,80){\line(1,0){120}}
\put(0,0){\line(0,1){80}}
\put(120,0){\line(0,1){80}}
% Оси координат:
\put(40,25){\begin{picture}(40,40)%
      \put(20,0){\vector(0,1){40}}
      \put(0,20){\vector(1,0){40}}
      \put(40,22){\mathit{x}}
      \put(22,40){\mathit{y}}
    \end{picture}}
\end{picture}

```

Кстати говоря, размеры внутренней картинке можно было бы задать совершенно произвольно, например, (200, 200) или даже (0, 0) — команда `\put` бездумно размещает объекты таким образом, чтоб их точки отсчета имели указанные координаты, и при этом не интересуется, сколько места они реально занимают и не наложатся ли на текст или другие объекты.

5. Параметры, регулирующие вид псевдорисунка

Про один из таких параметров мы уже говорили — это единица измерения длин на псевдорисунке, обозначаемая `\unitlength`.

В какой-то мере можно регулировать и толщину линий на наших псевдорисунках. Для этого предусмотрены команды `\thinlines` (тонкие линии) и `\thicklines` (толстые линии). По умолчанию стоит режим, в котором линии будут тонкими. Команды `\thicklines` и `\thinlines` можно давать не только в преамбуле, но и в самом тексте (в том числе и внутри окружения `picture`, так что можно регулировать, какие линии будут толстыми, а какие тонкими). Если одна из этих команд дана внутри группы, то по окончании группы ее действие прекращается (не забывайте, что любое окружение само по себе образует группу).

Кроме того, можно задать произвольным образом толщину вертикальных и горизонтальных (но не наклонных!) линий. Для этих целей служит команда `\linethickness`. У этой команды один обязательный аргумент — толщина линий, выраженная в \TeX овских единицах длины. Если мы скажем

```
\linethickness{2.5mm}
```

то все вертикальные и горизонтальные отрезки на псевдорисунке будут иметь толщину 2,5 мм.

Глава VI.

Верстка текста с выравниванием

При работе на пишущей машинке печать таблиц, состоящих из нескольких колонок, не вызывает особых проблем: все литеры имеют одинаковую ширину, поэтому, если нужно напечатать колонку слов, идущих одно под другим, достаточно каждый раз делать одинаковое количество пробелов. Однако по-настоящему красивые и профессионально выглядящие шрифты (в частности, используемые \TeX ом) являются «пропорциональными» (каждая буква имеет свою ширину), а в этом случае добиться выравнивания в колонках сложнее.

В настоящей главе мы рассмотрим два основных способа, предоставляемых \LaTeX ом для верстки текста с выровненными колонками (например, таблиц). Начнем с менее мощного, но более простого — имитации табулятора.

1. Имитация табулятора

1.1. Элементарные средства

Табулятор имитируется в \LaTeX е с помощью окружения `tabbing`. При верстке таблиц с помощью этого окружения пользователь сам задает места, в которых должна начаться очередная колонка. Конкретно это выглядит так. При наборе первой строки этого окружения можно в любой момент поставить команду `\=` — она отмечает очередное место, с которого начинается новая колонка («позицию табулятора», как на пишущей машинке). \LaTeX это место (расстояние от начала строки) запоминает. В дальнейшем можно с помощью команды `\>` «перескочить» к очередной позиции табулятора — 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\quad\=конец\\</code>
раз	два	три	<code>раз\>два\>три\\</code>
начинаем	продолжаем	заканчиваем	<code>начинаем\>продолжаем\></code>
			<code>заканчиваем\\</code>
			<code>\end{tabbing}</code>

В первой строчке мы задали две позиции табуляции двумя командами `\=` (на всякий случай мы разделили слова в первой строке и, тем самым, наши позиции табуляции, дополнительными пробелами — отсюда команды `\quad`). Первая строка завершается командой `\\`, а во

второй строчке мы уже начинаем пользоваться установленными позициями табуляции. Слово «раз» напечаталось с начала строки (каждая строка начинается с крайней левой позиции, если отсутствует команда вроде `\>`, задающая переход к новой позиции). Далее идет команда `\>` — «перейти на следующую позицию табуляции». И действительно, следующее после нее слово «два» начинается со второй позиции — как раз там же, где начиналось слово «середина». Перед словом «три» стоит еще одна команда `\>` — оно печатается с третьей позиции, как раз под словом «конец», с начала которого мы эту позицию и определили. Третья строчка 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 строчке слова при печати налезли друг на друга. Это и не удивительно: окружение `tabbing` исправно начинает очередную порцию текста с той позиции табуляции, которую мы ему укажем, но при этом отнюдь не проверяет, сколько места этот текст реально займет и не будут ли перекрываться колонки — за это целиком отвечает тот, кто текст готовит. Видимо, в данном случае следовало оставить побольше места при определении позиций табулятора (например, написать в первой строке `\qqquad` вместо `\quad`).

Кроме установки дополнительных интервалов экспериментальным путем, есть и другой способ правильно проставить позиции табулятора. Именно, если закончить строку не командой `\\`, а командой с суровым названием `\kill`, то эта строка не будет напечатана, но все позиции табулятора, установленные в ней, будут запомнены ЛАТЭХом, и их можно будет использовать в последующих строках. В вышеприведенном примере можно было бы написать так:

начало	середина	конец	<code>\begin{tabbing}</code>
раз	<i>два</i>	три	начинаем <code>\=</code> продолжаем <code>\=</code>
начинаем	продолжаем	заканчиваем	заканчиваем <code>\kill</code>
			начало <code>\></code> середина <code>\></code> конец <code>\\</code>
			<code>\bf раз\>\it два\>три\\</code>
			начинаем <code>\></code> продолжаем <code>\></code>
			заканчиваем <code>\\</code>
			<code>\end{tabbing}</code>

Для экономии места мы убрали из этой таблички лишнее «раз, два, три»; помимо этого, обратите внимание, что при установке позиций табулятора 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 табулятора не была впрытык к концу слова «начинаем»). Заметьте также, что во второй строчке мы убрали команды `\quad`; можно было бы их и оставить — на внешний вид таблицы это бы никак не повлияло, поскольку позиции табулятора уже установлены и лишние пробелы перед очередной командой `\>` никого не волнуют. По этой же причине мы не потрудились оставить пробелы между словами и `\>` в последней, предназначенной для печати строчке «начинаем, продолжаем, заканчиваем». Наконец, обратите внимание и на то, как мы меняли шрифт в строчке «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 внимательный читатель, обычно имеет совсем другой смысл — постановка диакритического знака над буквой (см. таблицу на стр. 63). Команды `\'` и `\`` также имеют внутри этого окружения особый смысл, о котором пойдет речь дальше. Поэтому, если внутри `tabbing` нам понадобился диакритический знак (скажем, над буквой *e*), то надо руководствоваться такой таблицей:

Внутри окружения `tabbing`

вместо надо набирать

`\=e` `\a=e`

`\'e` `\a'e`

`\'e` `\a'e`

1.2. Более сложные средства

А. Интервалы и разрывы между строками. Команда `\\` внутри окружения `tabbing` может иметь необязательный аргумент, действующий формально так же, как для этой команды, употребляемой внутри абзаца: если в квадратных скобках поставить длину (измеренную в воспринимаемых Т_EXом единицах, см. раздел I.2.10, или же какой-либо Л^AT_EXовский 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.

Б. Переустановка позиций табуляции. Команды `\=`, устанавливающие позиции табуляции, можно давать не только в первой строке. Опишем точно, как эта команда взаимодействует с `\>`.

Внутри окружения `tabbing` в каждый момент Л^AT_EXу 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 ноль (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 `\=`. 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 `\=`. Вот пример:

парочка позиций табуляции

 плюс еще одна здесь:

теперь их уже три

Вторую мы сменим и посмотрим:

где эти позиции теперь

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

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 `\roptabs`. Первая из них запоминает расположение позиций табуляции; после этой команды можно позиции переустановить, пользоваться этими новыми переустановленными позициями. . . — после команды `\roptabs` значения старых позиций табуляции будут восстановлены. Пример:

раз	два	три	четыре	<code>\begin{tabbing}</code>
гиппопотам	аллигатор	<code>раз\quad\=два\quad\=три\quad\=четыре\\</code>	<code>\pushtabs</code>	
раз	два	<code>гиппопотам\quad\=аллигатор\\</code>	<code>раз\>два\\</code>	
три	четыре	<code>три\>четыре\\</code>	<code>\roptabs</code>	
one	two	three	four	<code>one\>two\>three\>four\\</code>
un	deux	trois	quatre	<code>un\>deux\>trois\>quatre\\</code>
				<code>\end{tabbing}</code>

Команды `\pushtabs` и `\roptabs` должны быть «сбалансированы»: каждой `\pushtabs`, запоминающей позиции табуляции, должна соответствовать вспоминаящая их `\roptabs`. Если это условие не выполнено, Вы получите сообщение об ошибке.

В. 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 трогает»; просто весь текст, размещенный между `\>` или `\=` и `\'`, размещается левее позиции табуляции, определяемой командами `\>` 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>
à gauche	à droite	<code>\hspace{3.5cm}\=\kill</code>
links	rechts	<code>слева\>справа\'\'</code>
		<code>\a'a gauche\>\a'a droite\'\'</code>
		<code>links\>rechts\'</code>
		<code>\end{tabbing}</code>

Помимо прочего, обратите внимание, каким способом нам пришлось проставлять диакритический знак над а (см. стр. 123).

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 сверстать 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>links\ 'rechts\\</code>
		<code>\end{tabbing}</code>

Кстати, здесь нам вообще не понадобилось устанавливать позиции табуляции. Впрочем, смотрится эта таблица неважно.

Как мы уже отмечали, при начале новой строки текст начинается с нулевой позиции табуляции, то есть 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 табуляции, 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 `\-`). Нижеследующий пример иллюстрирует все эти изыски.

	<code>\begin{tabbing}</code>
раз два три четыре	<code>раз \=два \=три \=\kill</code>
два	<code>раз\>два\>три\>четыре\+\ \\</code>
три	<code>два\+\ \\</code>
четыре	<code>три\+\ \\</code>
три	<code>четыре\ \\</code>
четыре	<code>\<три\ \\</code>
два	<code>четыре\-\-\ \\</code>
раз два три четыре	<code>два\-\ \\</code>
	<code>раз\>два\>три\>четыре\ \\</code>
	<code>\end{tabbing}</code>

Описанные в этом разделе возможности окружения `tabbing` на практике используются редко, поскольку для верстки сложных таблиц в \LaTeX е есть более удобное средство — окружение `tabular`. Перейдем к его описанию.

2. Верстка таблиц

При пользовании окружением `tabbing` Вы должны самостоятельно следить чтобы разные колонки не накладывались друг на друга. Можно, однако, передать эти заботы программе: \TeX предоставляет возможности для верстки таблиц, в которых ширина колонок выбирается автоматически (по максимальной ширине их содержимого). В \LaTeX е для этих целей используются окружения `tabular` (для работы в «текстовых» режимах) и `array` (для работы в математическом режиме). Помимо автоматизированного определения ширины колонки, эти окружения дают возможность верстать разлинованные таблицы, таблицы, в которых некоторые записи охватывают несколько колонок, и т. д. В главе II уже шла речь про окружение `array`; здесь мы подробно разберем, как работает `tabular`; все возможности этого

окружения, о которых идет речь в этой главе, доступны и для `array`, и в последнем разделе главы 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 раздел II.3.2, то Вы уже заметили, что все это в точности совпадает с тем, что написано в этом разделе про окружение `array`. Разница лишь в том, что содержимое граф таблицы обрабатывается в окружении `tabular` как текст, а в окружении `array` — как формулы.

Прежде, чем мы начнем говорить о более сложных вещах, скажем о том, как окружение `tabular` взаимодействует с окружающим текстом. В отличие от других встречавшихся нам до сих пор окружений, оно *не* начинает печать с новой строки и *не* завершает текущего абзаца. Вся таблица, порождаяемая этим окружением, рассматривается Т_ЭXом 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 Т_ЭX сам позаботится о пробелах между таблицей и окружающим текстом.

А теперь посмотрите на 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>\tt itemize & нет\\</code>
<code>description</code>	нет	<code>\tt enumerate & есть\\</code>
		<code>\tt description & 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, что для этой команды внутри абзаца (стр. 74) или окружения `tabbing` (стр. 123): после строки вставляется дополнительный вертикальный промежуток (кстати, между строками таблицы, определенной с помощью окружения `tabular`, разрыв страницы *никогда* не происходит, так что в этом окружении у команды `\\` варианта «со звездочкой» нет). Во-вторых, команда `\tt` всякий

раз сменяла 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.

Л^AT_EX дает возможность сверстать и разлинованную таблицу. Для этого необходимо уметь задавать в таблице команды для создания горизонтальных и вертикальных отрезков («линеек» на полиграфическом жаргоне — см. раздел III.9). Горизонтальные отрезки задаются с помощью команды `\hline`. Эта команда может непосредственно следовать либо после `\\` (тогда отрезок печатается непосредственно после строки, завершенной этим `\\`), либо после `\begin{tabular}` с параметром (тогда отрезок печатается перед началом таблицы). Задаваемый командой `\hline` горизонтальный отрезок имеет ширину, равную общей ширине таблицы. Что касается вертикальных отрезков, то давайте для начала также ограничимся случаем, когда эти отрезки, разделяющие колонки таблицы, простираются на всю ее высоту, сверху донизу. Такие отрезки проще всего предусмотреть в преамбуле таблицы. До сих пор мы говорили, что преамбула таблицы — это последовательность из букв `l`, `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 вертикальный отрезок, разделяющий эти колонки. Можно поставить символ `|` перед первой из этих букв или после последней — тогда вертикальная линия 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 линией. Вот пример разлинованной таблицы:

7CA	шестнадцатеричная
3712	восьмеричная
11111001010	двоичная
1994	десятичная

```

\begin{tabular}{|l|l|}
\hline
7CA & шестнадцатеричная \\
\hline
3712 & восьмеричная \\
\hline
11111001010 & двоичная \\
\hline
1994 & десятичная \\
\hline
\end{tabular}

```

Две команды `\hline` могут следовать одна непосредственно после другой; в этом случае на печати получатся два горизонтальных отрезка, один под другим, разделенные по вертикали небольшим интервалом. Если слева и справа таблица ограничена вертикальными отрезками, то на пересечении крайних вертикальных отрезков с горизонтальными на печати получится разрыв:

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

```

\begin{tabular}{|c|c|}
\hline
Северо-Запад & Северо-Восток \\
\hline
Юго-Запад & Юго-Восток \\
\hline
\end{tabular}

```


Позже мы расскажем, как можно избавиться от этого полиграфического недостатка.

2.2. Более сложные случаи

А. Надписи, охватывающие несколько колонок. Чтобы создать такую надпись, нужно на месте соответствующей графы таблицы записать команду `\multicolumn`. У этой команды три обязательных аргумента:

- 1) Количество колонок, охватываемых нашей «нестандартной» графой.
- 2) «Преамбула» нашей графы. В качестве таковой может выступать буква `l`, `r` или `c` (если мы хотим, чтобы текст в графе был прижат влево, вправо или центрирован соответственно), возможно, с символами `|` слева или справа, если мы хотим, чтобы графа была ограничена вертикальными линиями.
- 3) Текст, записываемый в графу.

Если таблица, в которой Вы используете `\multicolumn`, является к тому же еще и линованной, то возможностей команды `\hline` для рисования горизонтальных отрезков может не хватить: иногда бывает нужен горизонтальный отрезок, простирающийся не на всю ширину таблицы, а охватывающий только часть ее колонок. Для рисования таких отрезков предусмотрена команда `\cline`. Как и `\hline`, ее нужно давать сразу после `\\`, но она имеет обязательный аргумент — номера первой и последней из колонок, охватываемых горизонтальной чертой, разделенные знаком «минус». Вот пример, в котором используются обе вышеописанные команды.

Западные сладости		
Сникерс	штука	330
	десяток	3000
Марс	штука	270
Баунти	штука	350
Твикс	В продаже	
Виспа	сегодня нет	

Получается эта таблица таким образом:

```
\begin{tabular}{|l|lr|}
\hline
\multicolumn{3}{|c|}{Западные сладости}\\
\hline
Сникерс & штука & 330\\
\cline{2-3}
& десяток & 3000\\
\hline
Марс & штука & 270\\
Баунти & штука & 350\\
\hline
Твикс & В продаже & \\
\cline{1-1}
Виспа & сегодня нет & \\
\hline
\end{tabular}
```

Б. Абзацы в графах таблицы. Иногда требуется, чтобы в графе таблицы стола не строка, а сверстаный абзац текста. Чтобы этого добиться, надо в преамбуле вместо буквы `l`, `s` или `r`, описывающей структуру колонки, написать `r{...}`, где вместо многоточия должна быть указана ширина колонки (в `TeX`овских единицах — см. стр. 16). Вот, например, 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.

А вот как задать эту таблицу в `LaTeX`:

```
\begin{tabular}{|p{5cm}|p{5cm}|}
\hline
\multicolumn{2}{|c|}{\large\bf 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}
```

В. Что такое колонка? При работе с линованными таблицами возникает следующий вопрос: как, собственно говоря, `LaTeX` понимает слова «одна колонка»? Пусть, например, преамбула таблицы имеет вид `||c|l|l||r|l|l|`, и мы в одной из строк написали, скажем,

```
Что-то&\multicolumn{1}{r}{Что-то еще}&&&И еще&Еще\}
```

Спрашивается, напечатается ли в этой строке вертикальная черта 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); спрашивается, сколько вертикальных черточек будет напечатано в конце этой строки: две, одна или ни одной? Ответ таков. `LaTeX` разделяет преамбулу на части, соответствующие колонкам. Если в преамбуле присутствуют только буквы `l`, `s`, `r` или `p` (последняя, разумеется, в составе выражения `r{...}`), то каждая такая часть — это просто соответствующая буква. 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`, `s`, `r` или `p` (последняя — вместе со всем выражением `r{...}`).

- Каждая колонка, кроме первой, начинается с буквы.

В вышеприведенном примере, в частности, колонки устроены так:

Преамбула	c l l r l l
Первая колонка	c
Вторая колонка	l
Третья колонка	l
Четвертая колонка	r
Пятая колонка	l
Шестая колонка	l

Поэтому в конце строки таблицы с такой преамбулой, оборванной после третьей колонки, будут напечатаны две вертикальные черты, поскольку они третьей колонке принадлежат. А если на месте второй графы такой таблицы написано `Что-то&\multicolumn{1}{r}{Что-то еще}`, то вертикальная черта между первой и второй графой также будет напечатана: эта черта является принадлежностью первой колонки, и команда `\multicolumn`, меняющая оформление второй графы, отменить ее не может.

3. Примеры

В этом разделе мы приведем различные примеры верстки сложных таблиц с помощью \LaTeX . 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 главе II.

Наш первый пример — таблица французских притяжательных местоимений, взятая из русско-французского словаря акад. Л.В. Щербы:

	существительные формы	прилагательные формы
мой	{ le mien, la mienne les miens, les miennes	mon, ma, mes
твой	{ le tien, la tienne les tiens, les tiennes	ton, ta, tes
его, ее, свой	{ le sien, la sienne les siens, les siennes	son, sa, ses
наш	le nôtre, la nôtre, les nôtres	notre, nos
ваш	le vôtre, la vôtre, les vôtres	votre, vos
их, свой ¹	le leur, la leur, les leurs	leur, leurs

¹Лишь в значении принадлежности 3-му лицу.

Вот как мы задали эту таблицу в \LaTeX :

```
{\small
\begin{tabular}{c|l}
\multicolumn{2}{c}{существительные формы}
& прилагательные формы\\[5pt]
мой & $\left\{
\begin{tabular}{l}
le mien, la mienne
les miens, les miennes
\end{tabular}
\right.
```

```

        le mien, la mienne\\
        les miens, les miennes\\
    \end{tabular}
    \right.
$
    & mon, ma, mes\\[8pt]
твой & $\left\{
    \begin{tabular}{l}
        le tien, la tienne\\
        les tiens, les tiennes\\
    \end{tabular}
    \right.
$
    & ton, ta, tes\\[8pt]
его, ее, свой &
    $\left\{
    \begin{tabular}{l}
        le sien, la sienne\\
        les siens, les siennes\\
    \end{tabular}
    \right.
$
    & son, sa, ses\\[8pt]
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}{\textsuperscript{$^1$}\rule{0pt}{11pt}\footnotesize
    Только в значении принадлежности 3-му лицу.}
\end{tabular}%
}

```

Разберем, как устроена эта таблица. Как явствует из ее преамбулы `cll`, она состоит из трех колонок, из которых левая центрирована, а две другие прижаты влево. Соответственно, три строки, начинающиеся со слова «наш», набраны совершенно бесхитростно. Заголовок таблицы сделан с помощью команды `\multicolumn`; от следующей графы в этой строке она отделена знаком `&`. Команда `\\`, завершающая первую строку таблицы, имеет у нас необязательный аргумент; это сделано, чтобы отодвинуть заголовок на 5 пунктов по вертикали от остальной части таблицы.

Рассмотрим теперь, как устроена вторая строка (начинающаяся с местоимения «мой»). Запись

$$\left\{ \begin{array}{l} \text{le mien, la mienne} \\ \text{les miens, les miennes} \end{array} \right.$$

образует в нашей таблице одну графу. Для того, чтобы получить фигурную скобку требуемого

(и неизвестного нам заранее) размера, мы воспользовались командами `\left` и `\right`, применяемыми при наборе формул (см. раздел II.2.5). Так как эти команды вне формул использовать нельзя, нам пришлось оформить эту графу таблицы 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 `array` (с преамбулой 1). Команды `\l`, завершающие первые три строки основной части таблицы, имеют необязательные аргументы, задающие дополнительные вертикальные пробелы после этих строк. Если не задавать этих необязательных аргументов, то фигурные скобки будут упираться друг в дружку и портить вид таблицы. Конкретные величины дополнительных пробелов были подобраны опытным путем.

К местоимению «свой» в последней строке таблицы дана сноска. Знак сноски реализован нами опять же как математическая формула — верхний индекс 1 к «пустой формуле»; текст сноски реализован как графа таблицы, охватывающая все три колонки (с помощью команды `\multicolumn`). Команда `\footnotesize` задает размер шрифта, используемый в обычных сносках (см. раздел III.4). 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}{9pt}
```

Как объясняется в разделе III.9, 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.

Остается заметить, что вся таблица в целом набрана мелким шрифтом, поскольку при шрифте нормального размера таблица не поместилась бы по ширине на страницу.

Следующий пример таблицы — расписание работы химчистки (вымышленной):

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

Л^AT_EXовский исходный текст для этого расписания выглядит так:

```
\begin{tabular}{lr@{--}l@{\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 палочки `|`, обозначающие разделительные вертикальные линии. Это, однако, — не вся правда. В качестве разделителя колонок (а также с краев) в преамбуле может быть использовано еще и так называемое «at-выражение»¹: символ `@`, непосредственно после которого в фигурных скобках записан какой-то текст, возможно, с `TeX`овскими командами. В таблице этот текст будет вставлен между соответствующими колонками во всех строках (если, разумеется, формат какой-то графы таблицы не был изменен командой `\multicolumn`). Мы использовали at-выражение трижды: два раза для вставки тире и один раз — для слова «Обед». Возникает вопрос, зачем нам понадобились команды `\qqquad` и `\quad` вокруг этого слова? Дело в том, что между колонками, разделенными at-выражением, *не* вставляется дополнительный интервал, которым `LaTeX` 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` шесть раз 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 (иногда это помогает, 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. раздел I.2.6 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 расстоянием между колонками, можно менять толщину линеек в линованных таблицах (обозначается `\arrayrulewidth`; относится этот параметр как к `array`, так и к `tabular`), а также расстояние между соседними линейками — это расстояние обозначается `\doublerulesep`, и оно также относится в равной мере к `array` и к `tabular`.

Теперь разберем обещанный пример, в котором приходится менять заданное по умолчанию расстояние между колонками. Наш пример относится к делению многочленов в столбик. Посмотрите на такую формулу:

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

Она была создана с помощью следующих `LaTeX`овских команд:

\$\$

¹Мы выбрали для него такое название, поскольку официально символ `@` называется «коммерческое at»; неофициально его называют самыми разными именами, от «собаки» до «блямбы».

```

\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 строчке выходили непомерно большими. А теперь разберем исходный текст поподробнее. 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 палочка в преамбуле перед буквой `r`, задающей пятую колонку — для вертикальной черты, входящей 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 вертикальная черта не пошла ниже, чем нужно: без нее, с преамбулой `rrr|r`, вертикальная черта относилась бы к четвертой колонке (в соответствии с правилами на стр. 129), и в результате третья строка закончилась бы вертикальной чертой, что нам совсем ни к чему.

Осталось заметить, что пары долларов, ограничивающие выключную формулу, заодно ограничивают и группу, так что по окончании формулы закончится и группа, и старое значение `\arraycolsep` восстановится автоматически.

Наш последний пример использования окружения `tabular` связан с проблемой, с которой мы столкнулись на стр. 127: как ликвидировать разрыв в вертикальных линейках, получающийся, если в линованной таблице написать две команды `\hline` подряд? Первое, что приходит в голову — создать еще одну строку в таблице, в которой поместить только невидимую линейку высотой, скажем, 2 пункта; казалось бы, тогда горизонтальные линейки будут на расстоянии 2 пункта друг от дружки, а вертикальные линейки не будут прерываться. Результат, однако, получается совершенно неудовлетворительный:

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

```

\begin{tabular}{|c|c|}
\hline
Северо-Запад & Северо-Восток\\
\hline
\rule{0pt}{2pt}&\\
\hline
Юго-Запад & Юго-Восток\\
\hline
\end{tabular}

```

Чтобы понять, в чем тут дело, нам придется обсудить, каким образом \LaTeX собирает таблицу из отдельных строк.

Таблицы, созданные с помощью окружения `tabular` или `array`, собираются из отдельных строк, которые вплотную приставляются друг к дружке. 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 \TeX у не помогла: ведь `\strut` все равно выше! Чтобы обойти эту трудность, в \TeX 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 главе VII, а пока что 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 нужного нам эффекта; надо только не забыть поставить в нужные строки команду `\strut` в явном виде, коль скоро автоматически это теперь не делается:

Северо-Запад	Северо-Восток
Юго-Запад	Юго-Восток

```
{\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, но тем не менее). Закрывающая фигурная скобка в последней строке закрывает группу, из которой была дана команда `\renewcommand`.

Если строк 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 линия 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.

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} .
```

Можно также вместо десятичной запятой поставить десятичную точку.

4. Заключительные замечания

Итак, мы познакомились с двумя способами верстки таблиц с помощью \LaTeX : окружениями `tabbing` и `tabular`. Какой из них лучше?

Окружение `tabbing`, по сравнению с `tabular`, работает быстрее и занимает меньше памяти: каждая строка таблицы, созданной с помощью `tabbing`, обрабатывается \TeX ом по отдельности и один раз, в то время как при работе окружений `tabular` и `array` \TeX сначала прочитывает всю таблицу до конца, затем определяет максимальную ширину каждой колонки, и лишь после этого начинается собственно верстка. Помимо замедления работы, это может привести и к тому, что для обработки некоторых особенно сложных и обширных таблиц \TeX у вообще не хватит памяти (в моей практике так случилось; до сих пор проблеме удавалось решить, обработав тот же текст с помощью «большой» версии \TeX а, лучше всего — на компьютере с 80386 или более мощным процессором). Еще одно преимущество окружения `tabbing` — то, что таблица, созданная с его помощью, может разместиться на нескольких страницах, в то время как таблица, созданная с помощью `tabular` или `array`, не может ни простираться на несколько страниц, ни быть перенесенной с одной страницы на другую.

С другой стороны, при пользовании окружениями `tabular` или `array` Вам не надо ломать голову, где разместить позиции табуляции, чтобы колонки не налезали одна на другую: \TeX сам сделает все необходимые вычисления. Тем более неоспоримо преимущество этих окружений, если таблица должна быть линованной или иметь сложную структуру. Кстати говоря, то обстоятельство, что таблица, созданная с помощью `tabular`, не может переноситься через страницу, — недостаток только \LaTeX а, но не \TeX а; с помощью \TeX овской команды `\halign` можно создавать таблицы, в которых и размеры колонок вычисляются автоматически, и переходы через страницы допускаются. В \LaTeX е этой командой пользоваться также можно, но описывать здесь, как она работает, мы не будем. Смелый читатель может попробовать прочитать об этом самостоятельно в книге [2].

В некоторых случаях верстку с выравниванием можно делать вообще «вручную», без помощи `tabbing` или `tabular`. Некоторые примеры тому мы увидим в главе VIII.

Глава VII.

Создание новых команд

Средства \LaTeX а, описываемые в этой главе, позволяют значительно сократить писанину при верстке сложных текстов. Именно, мы расскажем, как создавать новые команды (или, если угодно, сокращенные обозначения), заменяющие собой длинные фрагменты из текста и \TeX овских команд. По-ученому 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 главы II, нетрудно понять, что генерируется этот значок внутри математической формулы такой последовательностью команд:

```
\stackrel{\rm def}{=}
```

Часто писать такой длинный набор команд утомительно. Вот бы в \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{\rm def}{=}}
```

После того, как \TeX прочтет эту строчку, он всюду, встречая команду `\eqdef`, будет реагировать точно так же, как если бы он видел текст `\stackrel{\rm def}{=}`. Например, чтобы получить формулу $x^2 \stackrel{\text{def}}{=} x \cdot x$, теперь достаточно написать

```
x^2\eqdef x\cdot x
```

Новая команда \TeX а, которую мы определили, называется макросом (синонимы: макроопределение, макрокоманда). Рассмотрим теперь 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. раздел I.2.3): либо `backslash` и после него одна не-буква, либо `backslash` и после него — последовательность букв. Второй обязательный аргумент команды `\newcommand`, называемый

«замещающим текстом», сообщает Т_ЭXу 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. Впрочем, если Вы и попытаете так сделать, Л^AT_ЭX Вам этого не позволит, выдав сообщение об ошибке. Если Ваша русификация Т_ЭXа позволяет использовать в именах макросов русские буквы, имеет смысл этим воспользоваться: при этом резко снизится вероятность нарваться на запрещенное имя команды, не говоря уж о том, что русские имена более мнемоничны. Мы в дальнейших примерах также будем использовать русские буквы в именах макросов.

Если команда `\newcommand` дана внутри группы, то смысл определяемой ей новой команды будет забыт Т_ЭXом по выходе из группы. Если новая команда определяется в преамбуле, то, естественно, она будет понятна Т_ЭXу на протяжении всего документа.

В «замещающем тексте» макроопределения *не должно* присутствовать команд `\newcommand` (или `\renewcommand`, о которых пойдет речь ниже)¹. И еще одно ограничение, которое надо иметь в виду, создавая новые команды (макросы): во втором параметре команды `\newcommand` (иными словами, в «замещающем тексте») вместе с каждой открывающей фигурной скобкой должна присутствовать соответствующая ей закрывающая²: «несбалансированных» фигурных скобок в замещающем тексте быть не должно, так что макроопределения наподобие

```
\newcommand{\начатькурсив}{\it}
\newcommand{\кончитькурсив}{}}
```

являются незаконными и приведут лишь к сообщению об ошибке. Если Вам кажется, что такие ограничения стеснительны, можете изучить по книге [2], как их обходить; учтите, однако, что тогда Вам придется стать Т_ЭXником. Для большинства практических целей возможности по созданию макроопределений, предоставляемые Л^AT_ЭXом, вполне достаточны.

Еще одно ограничение: в замещающем тексте макроопределения нельзя пользоваться командой `\verb` или окружением `verbatim`.

Давайте теперь разберем несколько примеров, обращая внимание на типичные ошибки. Макросы хороши как средство скорописи, если Вам приходится сталкиваться со сложными наборами Т_ЭXовских команд в математических формулах. Их можно использовать и для совсем бесхитростных целей. Например, если в Вашем тексте часто встречается знак Δ , то Вам может надоест все время писать длинную команду `\bigtriangleup`. Коли так, придумайте сокращенное обозначение (скажем, `\btu`), напишите в преамбуле

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

и Вы сможете писать формулы наподобие

$$(A \Delta B) \cap C = (A \cap C) \Delta (B \cap C) \quad \$(A\btu B)\cap C= \\ (A\cap C)\btu (B\cap C)\$$$

На стр. 32 было рассказано, что делать, чтобы создать согласующееся с нашими традициями обозначение для функции тангенс. Теперь мы понимаем, что это был пример макроопределения (если Вам интересно знать, что значит команда `\mathop` в замещающем тексте, загляните в раздел II.4.4).

¹Если Вы понимаете, как можно было бы использовать такие экзотические макроопределения для дела, то Вам воистину пора читать Т_ЭXbook.

²Фигурные скобки, входящие в состав команд `\{` и `\}`, в счет при этом не идут

А вот пример, когда макрос заменяет длинную и сложную комбинацию из Т_ЕXовских команд (в этом примере мы предполагаем, что читатель знаком с содержанием раздела II.4.4). Итак, на стр. 57 мы объясняли, как получить формулу $E \underset{\neq}{\subset} F$, где $\underset{\neq}{\subset}$ — символ бинарного отношения. Давайте создадим для этого символа специальную команду. Придумаем для нее имя, скажем, `\subsetne` (по аналогии с `\subseteq`) и запишем в преамбуле:

```
\newcommand{\subsetne}{\mathrel
{\mathop{\subset}\limits_{\neq}}}
```

Теперь для получения формулы $E \underset{\neq}{\subset} F$ достаточно написать просто `E\subsetne F`.

В последних разделах главы II Вы найдете массу других примеров громоздких конструкций, для которых имеет смысл создать макросы. Практически польза от макроса начинает ощущаться, если конструкция, которую он заменяет, встречается в тексте не меньше четырех-пяти раз.

А вот 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 `^` преступно употреблен вне математической формулы. Удивительного в этом нет: Т_ЕX старательно подставляет вместо `\Pn` тот «замещающий текст», который Вы ему сообщили во втором аргументе команды `\newcommand`. В результате этого при развертывании макроса `\Pn` текст `$x \in \Pn$` превращается в незаконный текст `$x \in \mathbf{P}^n$`, в котором математическая формула заканчивается со вторым из знаков доллара, а символ `^` оказывается посреди обычного текста. Чтобы можно было напечатать \mathbf{P}^n не только изолированно, не надо включать знаки доллара в определение:

```
\newcommand{\Pn}{\mathbf{P}^n}
```

При этом придется, конечно, ставить знаки доллара вокруг `\Pn` в тех случаях, когда в тексте встречается просто \mathbf{P}^n , но зато наш макрос можно будет использовать и как составную часть более сложных формул.

Создавать макросы полезно не только для сокращения числа нажатий на клавиши при наборе формул. Вот пример, когда макросы помогают и при наборе обычного текста. Предположим, в нашем тексте много задач, причем условие каждой из задач начинается новый абзац (как обычно и бывает). Предположим также, что эти задачи никак не нумеруются (это предположение, в отличие от предыдущего, малореалистично, но оно сделано только для простоты и временно; в дальнейшем мы узнаем, как сделать так, чтоб Л_АT_ЕX сам нумеровал задачи для нас). 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 `{\bf ...}`, а просто написать `\z`. Первым обычно приходит в голову что-нибудь такое:

```
\newcommand{\z}{\bf 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: Т_EX опять пунктуально заменил \z на «замещающий текст», в результате чего получилось вот что:

```
\bf Задача. Пять парней . . .
```

Команда \bf оказалась *не* внутри группы, и весь текст пошел 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}{\bf Задача}}
```

На сей раз \z будет заменяться на {\bf Задача} , чего мы и хотели (при разворачивании макроса отбрасывается внешняя пара фигурных скобок, ограничивающая второй аргумент команды \newcommand, и только она!). Впрочем, наш макрос \z еще не идеален. Во-первых, точка после слова, напечатанного жирным шрифтом, выглядит лучше, когда и она сама набрана жирным шрифтом, поэтому точку тоже разумно 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 организовать, написав

```
\z{} Пять парней . . .
```

но лучше вставить и пробел прямо в определение:

```
\newcommand{\z}{\bf 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 (стр. 78):

```
\newcommand{\z}{\par{\bf Задача. }}
```

Если перед нашим макросом `\z` пустая строка все-таки будет, то лишняя команда `\par` ни на что не повлияет (см. стр. 78 и ниже), а если ее не будет, то `\par` сыграет роль недостающей пустой строки.

После того, как какой-то элемент текста оформлен с помощью макроса, становится удобно и менять это оформление. Например, если вдруг понадобилось оформлять все задачи так, чтобы соответствующие абзацы начинались без отступа, достаточно внести небольшое изменение в наше определение команды `\z`:

```
\newcommand{\z}{\par\noindent{\bf Задача. }}}
```

Если бы каждую задачу мы оформляли вручную, нам пришлось бы перед каждым словом «**Задача**» вписывать команду `\noindent`. В дальнейшем, когда мы познакомимся с ЛАТЭХовскими «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 главе IV: если мы хотим, чтобы в заголовках глав печаталось именно слово «Глава», а не «Chapter», то необходимо определить по-новому команду `\chaptername`. Для такого рода целей используется команда `\renewcommand`. Она устроена точно так же, как `\newcommand`, с тем отличием, что в качестве ее первого аргумента надо указывать имя *уже существующей* команды; в этом случае по выполнении команды `\renewcommand` значение этой команды изменится: она превратится в сокращенное обозначение для текста, указанного в качестве ее второго аргумента. Например, если написать

```
\renewcommand{\alpha}{Ку-ку}
```

то команда `\alpha` будет генерировать не то, что обычно (букву α в математической формуле и сообщение об ошибке, если эта команда употреблена вне математической формулы), а текст «Ку-ку».

Если команда `\renewcommand` дана внутри группы, то созданное ею переопределение значения команды забудется по выходе из этой группы. Если в качестве первого аргумента команды `\renewcommand` указать имя несуществующей команды, то Вы получите сообщение об ошибке.

Команда `\renewcommand` при неаккуратном обращении может привести к неприятностям. Дело в том, что нередко команды ЛАТЭХа определяются в терминах других команд ЛАТЭХа, причем знать эти сложные связи рядовой пользователь вовсе не обязан. На практике такое незнание может привести к тому, что безобидное на первый взгляд использование в своих целях имени какой-нибудь команды, которая в Вашем тексте ни разу не встречается, вызовет необъяснимо неправильную работу других команд. Поэтому используйте `\renewcommand` только тогда, когда Вы полностью отдаете отчет в своих действиях.

Выше мы часто называли макросы просто командами, и это — не просто небрежность речи: на самом деле принципиальной разницы между ЛАТЭХовскими командами и макросами почти нет. В частности, почти все команды ЛАТЭХа, 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 ЛАТЭХу к моменту запуска программы, так что определять их каждый раз не нужно. Макросы, в свою очередь, могут ссылаться на другие макросы (кстати, команды `\stackrel` и `\cdot`, на которые ссылалась наша команда `\eqdef`, также являются макросами). . . — и так далее, пока не дойдет до так называемых «примитивных команд» ЛАТЭХа. Из команд, о которых мы Вам рассказывали, примитивными являются считанные единицы: `\left` и `\right`, `\par`, `\-` и некоторые другие.

1.2. Команды с аргументами

В предыдущем разделе мы научились создавать новые команды, не требующие аргументов. Но мы хорошо знаем, что многие ЛАТЭХовские команды принимают аргументы, и часто возникает потребность создать новую команду с такими возможностями. Пусть, например, в Вашем тексте часто встречается «символ Лежандра» (если Вы не знаете, что это такое — не страшно), выглядящий так: $\left(\frac{a}{p}\right)$. Для получения этого символа в исходном тексте надо написать (внутри формулы, естественно) так:

$$\backslash\text{left}\{\backslash\text{frac}\{a\}\{p\}\backslash\text{right}\}$$

Хорошо бы создать команду `\символ` с двумя аргументами, чтобы можно было написать в формуле `\символ{a}{p}` и получить на печати $\left(\frac{a}{p}\right)$. Что ж, ЛАТЭХ предоставляет нам возможность сделать и это. Для создания команды с аргументами используется все та же команда `\newcommand`, но с необязательным аргументом. Посмотрите, как можно определить команду `\символ`:

$$\backslash\text{newcommand}\{\backslash\text{символ}\}[2]\{\backslash\text{left}\{\backslash\text{frac}\{\#1\}\{\#2\}\backslash\text{right}\}\}$$

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 `\символ`. Например, если в формуле написать

$$\backslash\text{символ}\{a+b\}\{c\}$$

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 в макросе с аргументами 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, которое мы указывали в необязательном аргументе команды `\newcommand`, и каждый — в своей паре фигурных скобок (как обязательные аргументы любой другой ЛАТЭХовской 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.

Проиллюстрируем все сказанное примером. Пусть мы определили команду `\путаница` следующим образом:

$$\backslash\text{newcommand}\{\backslash\text{путаница}\}[4]\{К\#4к\#2\#1л\}$$

Тогда будет получаться, например, такое:

Крокодил Гена и его друзья.

$$\backslash\text{путаница}\{\text{ди}\}\{\text{о}\}\{\text{го}\}\{\text{ро}\}$$

Гена и его друзья.

В самом деле, первым аргументом команды `\путаница` будет `ди`, вторым `о`, третьим `го`, четвертым `ро`; при разворачивании появится сначала буква `К`, затем четвертый аргумент, затем буква `к`, затем второй, первый и наконец буква `п` — как если бы `Крокодил` присутствовал в исходном тексте собственной персоной.

Ни аргументы ЛАТЭХовских команд (в том числе и определенных Вами), ни «замещающий текст» в `\newcommand` не должны содержать «несбалансированных» (не имеющих пары) фигурных скобок.

В ЛАТЭХе многие команды могут принимать, наряду с обязательными, еще и необязательные аргументы. Не выходя за рамки ЛАТЭХа, создать новую команду с такими свойствами невозможно.

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 бы так:

Как мы уже отмечали
на стр.~\pageref{units}, ...

После первого десятка таких ссылок возникает желание сократить число нажатий на клавиши. В результате в преамбуле появилась строка

```
\newcommand{\стр}[1]{стр.~\pageref{#1}}
```

и ссылки на страницы стало возможно оформлять так:

Как мы уже отмечали
на \стр{units}, ...

Не говоря уж об экономии времени на набор команды `\pageref`, исходный текст, в котором сократилось количество латинских букв, стал более обозримым.

2. Счетчики

В этом разделе мы научимся самостоятельно организовывать автоматическую нумерацию, подобно тому, как ЛАТЭХ 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. ЛАТЭХ предоставляет возможность присваивать счетчику различные значения и изменять их, 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 ЛАТЭХовский счетчик имеет свое имя — последовательность букв (*без* знака \!). Прописные и строчные буквы в именах счетчиков различаются; если русские буквы можно употреблять в именах команд, то их можно употреблять и в именах счетчиков. Чтобы можно было работать со счетчиком, надо его создать командой `\newcounter`, имеющей один обязательный аргумент — имя, которое Вы придумали для счетчика. Например, команда

```
\newcounter{abcd}
```

создает новый счетчик с именем `abcd`. Если имя, которое Вы придумали для счетчика, уже занято (так может случиться, даже если команда `\newcounter` в Вашем тексте всего одна: некоторые счетчики в ЛАТЭХе уже определены в момент начала обработки Вашего текста), то ЛАТЭХ создавать счетчик с таким именем откажется и выдаст сообщение об ошибке.

В отличие от многих других ЛАТЭХовских команд, команда для создания нового счетчика является «глобальной»: даже если она давалась внутри группы, ЛАТЭХ не забудет о существовании определенного ей счетчика и после выхода из этой группы. Это отличает `\newcounter` от `\newcommand` и `\renewcommand`.

Что же можно делать со счетчиком? Во-первых, можно менять его численное значение (на программистском жаргоне: «присваивать счетчику различные значения»). При создании счетчика его значение устанавливается в 0; чтобы установить какое-то другое значение, используется команда `\setcounter`, имеющая два обязательных аргумента: первый — имя счетчика, второй — значение, которое счетчику присваивается. Если, например, написать

```
\setcounter{abcd}{1994}
```

то после того, как ЛАТЭХ прочтет эту команду, значение счетчика `abcd` установится равным 1994. Значение, которое присваивается счетчику, может быть и отрицательным, но обязано быть целым.

Другой способ изменить значение счетчика — это прибавить к нему какое-то целое число. Для этого используется команда `\addtocounter`. Эта команда также имеет два обязательных аргумента: первый — имя счетчика, второй — число, которое прибавляется к счетчику. Например, после выполнения команд

```
\setcounter{abcd}{100}
\addtocounter{abcd}{-27}
```

значением счетчика `abcd` будет число 73.

Команды, изменяющие значение счетчика, также являются «глобальными»: если с их помощью внутри группы значение счетчика было изменено, то по выходе из группы его прежнее значение *не* восстановится.

Главная задача ЛАТЭХа — не просто присваивать значения каким-то переменным, а верстать тексты, так что перейдем к самому главному: как выдать значение счетчика на печать. Самый распространенный случай — печать значения счетчика обычными («арабскими») цифрами. Для этого используется команда `\arabic`:

Шел по улице отряд — 40 маль-
чиков подряд.

```
\setcounter{abcd}{40}
Шел по улице отряд ---
\arabic{abcd}
мальчиков подряд.
```

Значение счетчика печатается текущим шрифтом: если значение счетчика равно, скажем, 1994, то, встретив команду `\arabic{abcd}`, Т_ЕX отреагирует так же, как если бы на ее месте в исходном тексте было написано 1994.

Чтобы напечатать значение счетчика римскими цифрами, надо воспользоваться командой `\Roman` (если мы хотим, чтобы римские цифры записывались прописными латинскими буквами) или `\roman` (чтобы записать римскую цифру строчными латинскими буквами):

Людовика XIV звали «Король-солнце».	<code>\setcounter{abcd}{14}</code>
	Людовика~\Roman{abcd} звали
	\лк Король-солнце\пк.

Естественно, при печати значения счетчика римскими цифрами это значение должно быть положительным числом.

Можно, наконец, напечатать букву латинского алфавита, порядковый номер которой равен значению счетчика. Для этого используются команды `\alph` (для печати строчной буквы) и `\Alph` (для печати прописной буквы):

Наконец я узнаю, какая буква стоит в латинском алфавите на седьмом месте! Вот она: g.	<code>\setcounter{abcd}{7}</code>
	Наконец я узнаю, какая буква
	стоит в латинском алфавите
	на седьмом месте!
	Вот она:~\alph{abcd}.

Если значение счетчика при пользовании этими командами превышает количество букв в латинском алфавите, то Л_АT_ЕX выдает сообщение об ошибке.

В Л_АT_ЕXе отсутствует команда, позволяющая напечатать *русскую* букву с номером, равным значению счетчика. Средствами Т_ЕXа такую команду нетрудно создать, что и сделано, например, в «русифицирующем стиле», использованном при написании этой книги.

Наконец, можно напечатать один из девяти символов, используемых иногда в 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.	<code>\setcounter{abcd}{0}</code>
	Для сносок в Англии
	применяют такие символы:
	<code> \$\addtocounter{abcd}{1}</code>
	<code> \fnsymbol{abcd}\$,</code>
	<code> \$\addtocounter{abcd}{1}</code>
	<code> \fnsymbol{abcd}\$,</code>
	<code> \$\addtocounter{abcd}{1}</code>
	<code> \fnsymbol{abcd}\$,</code>
	а дальше попробуйте сами.

Обратите внимание, как три идентичных фрагмента исходного текста дали на печати три разных символа.

Для полноты картины скажем об еще одной менее употребительной команде, связанной со счетчиками. Именно, в командах для изменения значений счетчиков можно вместо явного указания чисел использовать значения других счетчиков, для чего употребляется команда `\value`. Например, в результате выполнения команд

```
\newcounter{efgh}
\setcounter{abcd}{10}
\setcounter{efgh}{100}
\addtocounter{efgh}{-\value{abcd}}
```

значение счетчика `efgh` станет равно 90. Можно даже писать, например,

```
\setcounter{efgh}{1000}
\tolerance=\value{efgh}
```

но большого смысла в таких трюках, как правило, нет.

Давайте теперь применим наши познания о счетчиках для дела. На стр. 141 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 ЛАТЭХ проставит сам. Теперь мы в состоянии решить эту проблему. Во-первых, для этого надо создать счетчик, значение которого в каждый момент будет равно номеру последней обработанной задачи; во-вторых, в определении команды `\z` надо предусмотреть, чтобы всякий раз значение этого счетчика увеличивалось на единицу, а затем печаталось в качестве номера задачи. В качестве имени счетчика выберем бесхитрое `задача`:

```
\newcounter{задача}
```

(напомним, что при выполнении этой команды счетчику `задача` будет присвоено значение 0). Теперь модифицируем определение макроса `\z` так:

```
\newcommand{\z}{\par\addtocounter{задача}{1}%
{\bf Задача \arabic{задача}.} }
```

Напомним, что команда `\par` означает «завершить предыдущий абзац, если он еще не был завершен»; без нее можно обойтись, если мы будем ставить команду `\z` только после пустой строки. Знак процента мы поставили, чтобы убрать лишний пробел, порождаемый концом строки. Теперь при первом исполнении команды `\z` значение счетчика `задача` 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 `\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 уже упоминали, что к моменту начала обработки ЛАТЭХом 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 стилем предусмотрено разбиение на главы), `section`, `subsection` и т. д. При каждом исполнении, например, команды `\section` значение счетчика `section` увеличивается на 1, и значение этого счетчика в каждый момент равно номеру текущей секции. Поэтому, если в определении команды `\z` написать

```
\arabic{section}.\arabic{задача}.
```

то перед номером задачи будет печататься номер текущей секции и точка.

Но как же, все-таки, сделать, чтобы в каждой секции нумерация задач начиналась заново? Можно, конечно, после каждой команды `\section` присваивать счетчику `задача` значение 0 с помощью команды `\setcounter`, но это некрасиво и ненадежно (а вдруг забудем?). Вместо этого следовало бы сразу определить счетчик `задача` таким образом:

```
\newcounter{задача}[section]
```

При таком определении счетчик `задача` будет, как говорят, *подчинен* счетчику `section`: всякий раз, когда значение счетчика `section` увеличивается на единицу командой `\section`, значение счетчика `задача` будет устанавливаться в нуль, и тем самым счет задач будет в каждой секции начинаться заново. Стало быть, если считать, что счетчик `задача` определен именно так, как сказано выше, то очередной раз исправленное определение команды `\z` выглядит так:

```
\newcommand{\z}{\par\addtocounter{задача}{1}%
{\bf Задача \arabic{section}.\arabic{задача}.} }
```

При этом нумерация задач будет начинаться заново в каждом разделе (`section`), и вторая задача третьего раздела будет иметь номер 3.2.

Теперь сообщим точные правила создания счетчиков, подчиненных другому счетчику. Они просты: команда `\newcounter` может принимать один необязательный аргумент (*после* обязательного) — имя того счетчика, которому будет подчинен определяемый нами счетчик. Разумеется, в момент выполнения команды `\newcounter` с необязательным аргументом счетчик, имя которого дается в квадратных скобках, должен уже существовать.

Ко всему сказанному требуется одно важное уточнение: мы еще толком не объяснили, в каких случаях значение подчиненного счетчика устанавливается в нуль. В самом деле, пусть счетчик `слуга` подчинен счетчику `хозяин`; тогда команда

```
\addtocounter{хозяин}{1}
```

никоим образом не повлияет на значение подчиненного счетчика `слуга`: 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 счетчика, имя которого является ее аргументом, а значения всех счетчиков, подчиненных счетчику — ее аргументу, устанавливает в нуль. Если, например, в нашем тексте определены два счетчика так:

```
\newcounter{хозяин}
\newcounter{слуга}[хозяин]
```

то после выполнения команд

```
\setcounter{хозяин}{10}
\setcounter{слуга}{10}
```

значения обоих счетчиков станут равны 10, после выполнения команды

```
\addtocounter{хозяин}{1}
```

значение счетчика `хозяин` станет равно 11, а значение счетчика `слуга` не изменится, а вот после выполнения команды

```
\refstepcounter{хозяин}
```

значение счетчика `хозяин` станет равно 12, а значение счетчика `слуга` 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 стили Л^AT_EX_А. Например, если основным стилем является `book`, то 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 секций уже начинается заново в каждой главе.

2.3. Организация автоматических ссылок

Вернемся еще один, теперь уже последний, раз к нашей команде `\z`. Раз она автоматически нумерует задачи, то неплохо было бы, если б пронумерованные ей задачи можно было метить командой `\label` и ссылаться на эти метки командой `\ref` (проблема именно в ней, поскольку команда `\pageref`, дающая номер страницы, работает в любом случае). Если коротко, то решение этой проблемы таково: увеличивать на единицу значение счетчика `задача` надо не с помощью команды `\addtocounter`, которой мы пользовались до сих пор, а с помощью команды `\refstepcounter`, о которой уже шла речь по другому поводу в предыдущем разделе. Если мы определим команду `\z` так:

```
\newcommand{\z}{\par\refstepcounter{задача}%
{\bf Задача \arabic{section}.\arabic{задача}.} }
```

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 секции 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 ЛАТЭХовской конструкцией, связанной со счетчиками.

Мы уже знаем, что значение ЛАТЭХовского 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 секции называется `\thesection`, для вывода на печать номера главы — `\thechapter`, команды для вывода на печать определенных нами счетчиков `слуга` и `хозяин` — `\thesлуга` и `\thехозяин` соответственно. При создании счетчика автоматически определяется и соответствующая `the`-команда. Например, при создании счетчика `abcd` автоматически определяется команда `\theabcd` таким образом:

```
\newcommand{\theabcd}{\arabic{abcd}}
```

В дальнейшем эту команду можно переопределять:

Людовика XIV звали «Король-солнце».

```
\setcounter{abcd}{14}
\renewcommand{\theabcd}{%
{\Roman{abcd}}
Людовика~\theabcd звали
\лк Король-солнце\пк.
```

Мы же, чтобы при ссылках перед номером задачи печатался номер секции, в которой находится эта задача, и точка, переопределим команду `\theзадача` так:

```
\renewcommand{\theзадача}{\thesection.\arabic{задача}}
```

Если включить эту команду в преамбулу документа, то ссылки на сгенерированные нашей командой `\z` номера задач будут выглядеть должным образом.

В нашем переопределении команды `\theзадача` мы воспользовались командой `\thesection`, чтобы наши макросы правильно работали с любым основным стилем документа. Дело в том, что при разумном оформлении номер секции, предшествующий при ссылке номеру задачи, должен печататься таким же образом, как и номер секции при ссылке на секцию, а это в разных стилях делается по-разному: в стиле `article`, например, `\thesection` — это то же самое, что и `\arabic{section}` (иными

словами, ссылка на секцию, сгенерированная командой `\ref`, печатает просто номер секции), а в стиле `report` команда `\ref` при печати ссылки на секцию печатает не номер секции, а номер главы, точку и номер секции. Поскольку мы написали `\thesection`, все эти тонкости будут учтены автоматически.

Приведем еще один (игрушечный) пример использования счетчиков в макроопределениях, отчасти чтобы еще раз продемонстрировать применение подчиненных счетчиков, а отчасти — чтобы убедить читателя, что не боги горшки обжигают. Именно, давайте разработаем свои собственные команды для создания разделов документа, не полагаясь на `\chapter`, `\section` и т. п. из стандартных ЛАТЭХовских стилей. Поскольку пример игрушечный, мы сделаем ряд упрощающих предположений (в частности, предположив, что заголовки глав и разделов будут укладываться в одну строку) и не будем заботиться о том, насколько удачным выйдет оформление заголовков. Итак, приступим. Пусть наш документ делится на главы, которые, в свою очередь, делятся на разделы. Каждую главу будем начинать с новой страницы, перед каждым разделом оставлять 1 см (если, конечно, раздел не начинается новой страницы). Наконец, предусмотрим, чтобы главы и разделы автоматически нумеровались с возможностью создания автоматических ссылок на эти номера. Команды для создания главы и раздела назовем `\глава` и `\раздел` соответственно; это будут команды, требующие одного аргумента — названия главы или раздела.

Для того, чтобы номера глав и разделов генерировались автоматически, нам необходимо создать счетчики, содержащие эти номера. Пусть счетчик с номером главы называется `глава`, а счетчик с номером раздела — `раздел` (имена счетчиков могут совпадать с именами команд). Имея в виду, что нумерация разделов в каждой главе будет начинаться заново, напишем в преамбуле так:

```
\newcounter{глава}
\newcounter{раздел}[глава]
```

Теперь определим команду `\глава`:

```
\newcommand{\глава}[1]{\clearpage % с новой страницы
\vspace*{4cm}% оставить место сверху
\refstepcounter{глава}% новый номер главы
{\LARGE\bf % шрифт для заголовка
Глава \thеглава.\ #1% заголовок
\par % кончить заголовок
\vspace{5mm plus 1mm minus .5mm}% Промежуток между
% заголовком и текстом
}% завершить группу, внутри которой менялся шрифт
}% конец макроопределения
```

Поскольку очередной номер главы устанавливался командой `\refstepcounter`, при этом будет начата заново и нумерация разделов, а на номера глав можно будет делать автоматические ссылки с помощью `\label` и `\ref`. Обратите также внимание на команду `\thеглава`. Мы воспользовались ей, поскольку не хотим на этом этапе предрешать, в каком виде номер главы будет представлен в заголовке: в виде арабской цифры, римской цифры или еще как-нибудь. Если в дальнейшем нам захочется изменить вид этого представления, то нам не придется лезть, с риском ошибиться, в длинное определение команды `\глава`, а будет достаточно переопределить команду `\thеглава`. Заметьте, что промежуток между заголовком и текстом мы сделали растяжимым, чтобы помочь ТЭХу найти правильное разбиение на страницы (см. стр. 80). Наконец, `backslash` с пробелом после точки мы поставили, чтобы не увеличивать пробела между номером главы и ее названием (стр. 61).

Определение команды `\раздел` можно дать, например, так:

```
\newcommand{\раздел}[1]{\par % завершить предыдущий абзац
\pagebreak[2]\vspace{1cm plus 3mm minus .5mm}% см. ниже
\refstepcounter{раздел}% новый номер раздела
{\Large\bf % шрифт для заголовка
\therаздел{ } #1% заголовок
\par % кончить заголовок
}% завершить группу, внутри которой менялся шрифт
\nopagebreak % чтобы не оторвать текст от
% заголовка
\vspace{2mm plus 1mm}% Промежуток между заголовком
% и текстом
}% конец макроопределения
```

Пояснений тут требует команда `\pagebreak[2]`. Мы вставили ее в макроопределение, чтобы поменьше разделов начиналось внизу страницы. В самом деле, команда `\pagebreak[2]` предлагает \TeX у начать в этом месте новую страницу (см. раздел III.7.5Б); если так и будет сделано, то дополнительный вертикальный промежуток, созданный командой `\vspace`, пропадет, и заголовок раздела начнется с самого верха новой страницы; если же разрыва страницы все-таки не произойдет, то перед заголовком раздела будет вертикальный промежуток величиной 1 см (обладающий указанными в макроопределении растяжимостью и сжимаемостью).

Нам осталось только задать вид, в котором будут представляться на печати номера глав и разделов. Иными словами, надо переопределить должным образом команды `\theadглава` и `\theadраздел` (в момент создания счетчиков они, как мы помним, были автоматически определены таким образом, что `\theadглава` и `\theadраздел` — просто номер главы и раздела соответственно, набранный арабскими цифрами). Предположим, мы решили, что номера глав будут печататься римскими цифрами, а номер второго раздела четвертой главы будет иметь вид IV–2. Тогда требуемые переопределения таковы:

```
\renewcommand{\theadглава}{\Roman{глава}}
\renewcommand{\theadраздел}{\Roman{глава}--\arabic{раздел}}
```

Еще одно замечание: точки после номера главы мы включили в определение команды `\theadглава`, а не `\theadглава`, чтобы можно было пользоваться автоматическими ссылками: если бы `\theadглава` определялось как

```
\Roman{глава}.
```

то исходный текст

```
в главе \ref{метка} мы пишем...
```

дал бы на печати

```
в главе IV. мы пишем
```

что нелепо.

По сравнению с макроопределениями, реально используемыми в стандартных стилях \LaTeX а, мы в нашем игрушечном наборе макросов многого не предусмотрели: не позаботились ни о колонтитулах, ни об автоматически генерируемом оглавлении, внешний вид заголовков оставляет желать лучшего, и т. д. Тем не менее один из основных принципов у нас

присутствует: нумерация действительно организуется с помощью подчиненных друг другу счетчиков и команды `\refstepcounter`.

У внимательного читателя может возникнуть вопрос, каким образом команда `\label` узнаёт, какого вида ссылку ей генерировать. В самом деле, пусть исходный текст имеет вид

```
\chapter{Что-то}\label{a}
.....
\section{Кое-что}\label{b}
.....
\begin{figure}
\label{c}
.....
\end{figure}
```

и пусть глава оказалась по номеру третьей, помеченная секция — второй в этой главе, а помеченная плавающая иллюстрация — пятой в своей секции. Откуда \LaTeX знает, что команда `\ref{a}` должна сгенерировать просто цифру 3, команда `\ref{b}` — текст 3.2, а команда `\ref{c}` — текст 3.2.5? Ответ на этот вопрос таков: команда `\label` генерирует (т. е. записывает в `aux`-файл) ссылку в соответствии с видом того счетчика, который последним подвергнулся операции `\refstepcounter`. Поэтому в данном примере команда `\label{a}` генерирует ссылку на счетчик `chapter`, `\label{b}` — на счетчик `section`, а `\label{c}` — на счетчик `figure`, отвечающий за нумерацию плавающих иллюстраций.

2.4. Счетчики, которые определять не надо

Мы уже мельком упоминали, что при начале работы \LaTeX некоторые счетчики определены сразу. Например, это те счетчики, которые перечислены на стр. 148. Кроме того, заранее определен очень важный счетчик `page`, отвечающий за нумерацию страниц, а также счетчик `footnote`, ответственный за нумерацию сносок. Нумерацией плавающих иллюстраций и таблиц занимаются счетчики, называемые попросту `figure` и `table`. В приложении Г перечислены все эти заранее определенные счетчики и указано, каким счетчикам они подчинены (подчиненность иногда зависит от стиля 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. Модификация оформления перечней

Хороший пример переопределения команд доставляют перечни. В разделе III.8 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 такими галочками \surd , то достаточно написать в преамбуле

```
\renewcommand{\labelitemi}{\surd}
```

(команду `\surd` можно найти в таблице на стр. 35). Если окружение `itemize` расположено внутри другого окружения `itemize`, как в примере на стр. 86, то значки для пометки элементов перечня будут уже, вообще говоря, другими: их вид задается командой `\labelitemi`; вид значков для пометок элементов `itemize` на третьем и четвертом уровнях вложенности задается командами `\labelitemiii` и `\labelitemiv` соответственно; их также можно переопределять.

Теперь рассмотрим окружение `enumerate`. Коль скоро оно автоматически нумерует элементы перечня, можно предположить, что это окружение связано с ЛАТЭХовскими 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 соответственно. С другой стороны, сами значки, помечающие элементы перечня, порождаются командами `\labelenumi`, `\labelenumii`, `\labelenumiii` и `\labelenumiv` соответственно — в зависимости от уровня вложенности. Например, в стандартных ЛАТЭХовских стилях команда `\labelenumi` определена так:

```
\newcommand{\labelenumi}{\theenumi.}
```

в то время как `the`-команда, определяющая представление счетчика `enumi` на печати, определена просто как

```
\newcommand{\theenumi}{\arabic{enumi}}
```

Стало быть, если мы не меняем стандартного стиля, то элементы перечня `enumerate` (не вложенного в другой `enumerate`) будут нумероваться цифрами с точкой. Если же мы хотим, скажем, чтобы после цифры шла не точка, а скобка (как в нашей книге), то можно в преамбуле написать

```
\renewcommand{\labelenumi}{\theenumi)}
```

Если же мы к тому же хотим, чтобы элементы перечня нумеровались римскими цифрами, то можно написать еще и так:

```
\renewcommand{\theenumi}{\Roman{enumi}}
```

Аналогичным образом можно менять оформление нумерованных перечней на других уровнях вложенности.

Если Вы хотите изменить оформление перечня более серьезным образом (например, установить другую величину полей, или сделать так, чтобы значки, помечающие элементы перечня, были выровнены по левому, а не по правому краю), то Вам придется подождать до главы IX.

3. Параметры со значением длины

Наряду с со счетчиками — переменными с целочисленными значениями, при создании собственных макроопределений возникает нужда и в переменных, значениями которых являются длины. Например, в предыдущем разделе мы, разрабатывая команду `\раздел`, в явном виде задали промежуток между заголовком раздела и остальным текстом. Если этот промежуток нам почему-либо захочется изменить, то придется снова залезать в определение команды `\раздел`. Было бы удобнее, если бы в наше распоряжении был параметр под названием, скажем, `\отступ`, так что можно было бы в определении команды `\раздел` написать

```
\vspace{\отступ}
```

и потом отдельно написать, допустим,

```
\отступ=2mm
```

(или присвоить параметру `\отступ` другое значение). Правда, в богатом наборе \TeX овских и \LaTeX овских параметров требуемого нам параметра `\отступ` нет. Но это не страшно, поскольку его можно создать. Для этого используется команда `\newlength`:

```
\newlength{\отступ}
```

После того, как Вы, допустим, в преамбуле, дали эту команду, будет определен новый параметр со значением длины; его можно будет обычным образом использовать в аргументах команд наподобие `\vspace` и ему можно будет обычным образом присваивать значения.

Теперь — точные правила. Команда `\newlength` имеет один обязательный аргумент — имя команды, обозначающей определяемый Вами параметр. Это имя должно подчиняться обычным правилам для \TeX овских команд (backslash, после которого следует либо одна небуква, либо последовательность букв). Если это имя уже занято, \LaTeX выдаст сообщение об ошибке. Определение нового параметра, совершаемое командой `\newlength`, является «глобальным»: даже если эта команда была дана внутри группы, \TeX будет помнить о существовании этого параметра и по выходе из группы. По этой, в частности, причине разумное место для команды `\newlength` — преамбула.

Определенный нами параметр со значением длины приобретает такой же статус, как уже существующие \TeX овские и \LaTeX овские параметры, такие, как `\parindent`, `\textwidth`, и т. п. 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 разделе I.2.6 на примере параметра `\parindent`: для присваивания значения надо написать имя параметра, знак равенства, и после знака равенства — величину присваиваемой длины. Пробелы после указания единицы длины \TeX ом игнорируются (скорее всего, Вы будете присваивать значения параметрам в преамбуле документа или между абзацами, где лишние пробелы никого не волнуют). Длина должна быть выражена в единицах, воспринимаемых \TeX ом (см. их список в разделе I.2.10). Даже если Вы присваиваете нулевую длину, какая-то единица длины должна быть явно указана (например, `0pt`). Кроме того, можно воспользоваться \LaTeX овской командой `\setlength`, имеющей два обязательных аргумента: первый — имя параметра, второй — значение длины, присваиваемое этому параметру. Таким образом, команды

```
\parindent=1.5em
```

и

```
\setlength{\parindent}{1.5em}
```

равносильны. Наконец, отметим, что, как мы уже неоднократно отмечали, присваивания, сделанные внутри группы, забываются по выходе из этой группы.

В предыдущем абзаце мы немного лукавили, умолчав 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 Вам сообщение об ошибке, поскольку решит, что длина должна иметь, помимо «естественного размера», еще и `plus`- или `minus`-компоненту (см. стр. 80; ниже мы поговорим подробнее о такой возможности). Если Вы пишете текст

на русском языке, вероятность такого стечения обстоятельств ничтожна. Тем не менее, забывать о такой опасности не следует, и особенно если команда присваивания входит в макроопределение: Вы же не знаете заранее, в какое место вставите свой макрос. Чтобы застраховаться от этой неприятности раз и навсегда, пользуйтесь `\setlength`, хоть при этом и придется нажать на большее число клавиш. Ср. также обсуждение команд `\hrule` и `\vrule` в разделе III.9.

Параметры со значением длины можно использовать всюду, где в аргументе ЛАТЭХовской команды требуется указать размер. Пусть, например, в преамбуле документа написано

```
\newlength{\пример}
```

Тогда посмотрите на следующий пример:

9	9		<code>\пример=10mm</code>
8		8	<code>9\hspace{\пример}9</code>
9	9		<code>{\пример=20mm</code>
			<code>8\hspace{\пример}8}</code>
			<code>9\hspace{\пример}9</code>

Обратите внимание, что, если присвоение 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 `\пример` равно 10 мм, то команда `\hspace{2.71\пример}` сделает пробел длиной 27,1 мм.

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, ЛАТЭХ предоставляет еще полезную команду

```
\settowidth{параметр}{текст}
```

которая присваивает параметру *параметр* значение, равное ширине *текста*. Вот пример:

СЛОВО	слово	<code>\settowidth{\пример}{\Large слово }</code>
	слово	<code>{\Large слово }слово</code>
		<code>\hspace{\пример}слово</code>

В разделе III.7.3 у нас шла речь о том, что некоторые используемые в Т_ЕXе 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 `\пример` имел естественный размер 2 см и при этом мог растягиваться на 4 мм и сжиматься на один пункт, то можно написать так:

```
\setlength{\пример}{2cm plus 4mm minus 1pt}
```

4. Создание новых окружений

4.1. Новые окружения: общий случай

Как мы уже имели возможность убедиться, для сокращения времени на написание длинных последовательностей команд удобно пользоваться макросами. В тех случаях, когда для достижения необходимого нам эффекта требуется сложная последовательность команд в начале и в конце какого-то текста, ЛАТ_EX дает нам возможность создать оформить соответствующие макросы в виде нового окружения. Как это делается, разберем на примере.

Предположим, нам хочется взять в рамку абзац текста шириной 7 см. Один из возможных способов сделать это 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}
```

что даст на печати вот что:

<p>Этот текст будет заключен в рамку. Как видите, окружение, предназначенное для верстки таблиц, можно использовать и для этих целей.</p>

Если таких рамок с текстом у Вас много, то можно сократить число нажатий на клавиши, определив окружение с именем, скажем, `рамка`, так, чтоб можно было бы просто писать

```
\begin{рамка}
Этот текст будет ...
... этих целей.
\end{рамка}
```

Определяется это окружение так:

```
\newenvironment{рамка}{\begin{tabular}{|p{7cm}|}
\hline}{\\ \hline \end{tabular}}
```

В общем случае команда `\newenvironment` имеет такой формат:

```
\newenvironment{имя}{открывающие_команды}{закрывающие_команды}
```

Здесь *имя* — имя определяемого окружения, *открывающие_команды* — команды и/или текст, подставляемые вместо команды `\begin` с именем окружения, *закрывающие_команды* — команды и/или текст, подставляемые вместо команды `\end` с именем окружения.

Вместо окружения, определяемого с помощью `\newenvironment`, можно с тем же успехом создать два макроса: один для *открывающих_команд*, другой для *закрывающих*. Например, в нашем случае с рамкой можно было бы написать

```
\newcommand{\начать}{\begin{tabular}{|p{7cm}|}\hline}
\newcommand{\кончить}{\\ \hline \end{tabular}}
```

и создавать рамки так:

```
\начать
Этот текст...
\кончить
```

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 \LaTeX выдаст сообщение об ошибке, в котором ровно это Вам и скажет; если же Вы забудете команду `\кончить`, 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`, ... `#9`, причем эти значки можно употреблять *только в открывающих командах* (то есть во втором обязательном аргументе команды `\newenvironment`)

С помощью `\newenvironment` нельзя переопределить уже существующее окружение (если Вы все же попытаетесь так сделать, \LaTeX выдаст сообщение об ошибке). Если Вам действительно необходимо такое переопределение, надо пользоваться командой `\renewenvironment`, работающей точно так же, как и `\newenvironment`, с тем различием, что в качестве первого аргумента ей можно передавать только имя уже существующего окружения.

4.2. Окружения типа «теорема»

Если Вы пишете математический текст, то в этом тексте будет содержаться немало 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_ЕХовских команд, и лучше не повторять этот длинный набор команд много раз, а создать заменяющее его макроопределение, что, в свою очередь, может потребовать некоторого труда (чем-то подобным мы занимались в предыдущих разделах, когда разрабатывали команду \z). Если же Вы или редакция, с которой Вы имеете дело, не слишком требовательны к деталям оформления, то соответствующие макросы (точнее говоря, новые окружения) легко создать из полуфабрикатов, предоставляемых нам для этих целей Л^AT_EXом.

Окружения, используемые в Л^AT_EXе 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^AT_EX, в целях экономии машинной памяти и исходя из того, что на все вкусы таких окружений 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

Предложение 1. <i>Волга впадает в Каспийское море.</i>	<code>\begin{pred1}</code> Волга впадает в Каспийское море. <code>\end{pred1}</code>
Доказательство. См. любую географическую карту.	<code>{\bf Доказательство.} См. \</code> любую географическую карту.

Для создания такого окружения используется команда `\newtheorem`:

```
\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). <i>Пифагоровы штаны на все стороны равны.</i>	<code>\begin{pred1} [Пифагор]</code> Пифагоровы штаны на все стороны равны. <code>\end{pred1}</code>
--	---

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 `pred1`); если мы хотим изменить представление на печати номеров нашей «теоремы», то можно обычным образом переопределить соответствующую `the`-команду. Например, если мы хотим, чтобы предложения нумеровались прописными латинскими буквами, надо в преамбуле написать:

```
\renewcommand{\thepred1}{\Alph{pred1}}
```

«Теоремы», определяемые описанным выше способом, будут иметь сплошную нумерацию на протяжении всего документа. Как мы уже понимаем, это далеко не всегда удобно. Часто хотелось бы сделать так, чтоб, например, в каждой секции 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 каждой секции. Тогда можно написать в преамбуле так:

```
\newtheorem{theorem}{Теорема}[section]
```

После этого можно будет писать, например, вот что:

<p>Теорема 4.1. <i>Сумма углов треугольника равна 180°.</i></p>	<pre>\begin{theorem} Сумма углов треугольника равна 180°. \end{theorem}</pre>
--	---

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, Л^AT_EX 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[theorem]{lemma}{Лемма}
```

В этом случае необязательный аргумент команды `\newtheorem` располагается перед обязательными; этот аргумент — имя того окружения типа «теорема», совместно с которым будет нумероваться определяемая Вами «теорема».

В заключение остается отметить, что команду `\newtheorem` можно использовать или с одним необязательным аргументом, или с другим, но не с обоими вместе.

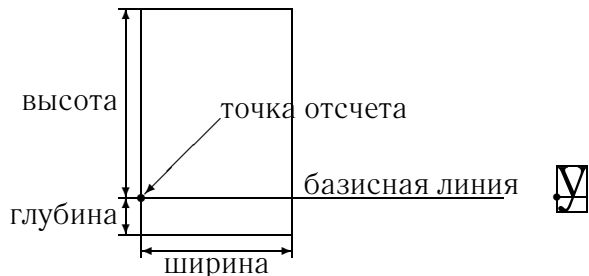
Глава VIII.

Блоки

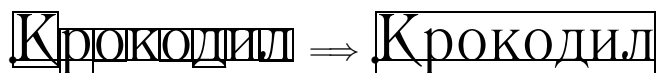
1. Текст состоит из блоков

Мы уже отмечали, что в процессе верстки Т_ЭХ 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.

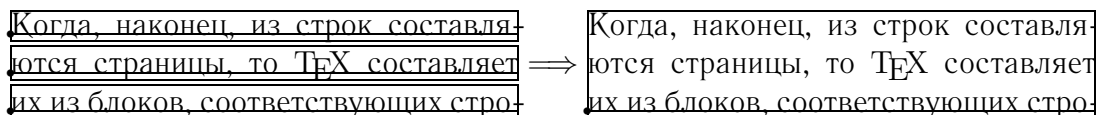
С точки зрения Т_ЭХа, каждая буква представляет собой *блок* (английский термин: box), то есть прямоугольник с выделенной *точкой отсчета*; горизонтальная прямая, проходящая через точку отсчета, называется *базисной линией* (английский термин: baseline). Блок характеризуется тремя размерами: шириной, высотой и глубиной. См. рисунок, на котором также изображен блок, соответствующий букве у.



Когда из букв составляется слова, а из слов — строки, то 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 крайнего левого из составляющих ее блоков:



Страницы — это тоже блоки. Когда они составляются из блоков, соответствующих строкам, то 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:



Подведем итоги. При верстке текста Т_ЭХ работает с блоками. Каждая буква также рассматривается как блок, но блоки могут состоять и из больших фрагментов текста. В приведенных выше примерах мы сталкивались с блоками, которые Т_ЭХ делает автоматически; в настоящей главе пойдет речь о командах, предназначенных для создания блоков вручную. Сначала мы расскажем, какие средства для этого предоставляет нам ЛАТЭХ, а затем рассмотрим Т_ЭХовские команды, имеющие больше возможностей, но более сложные.

2. ЛАТЭХовские команды для генерации блоков

2.1. Блоки из строк

С одной командой для генерации блоков мы уже знакомы: это команда `\mbox`. Эта команда создает блок из текста, набираемого в одну строку. Полученный блок рассматривается Т_ЭХом как одна большая буква:

Проказница	мартышка,	Проказница	мартышка,
осел, козел	и	<code>\mbox{осел, козел}</code>	и
косолапый		косолапый	мишка\ldots
мишка...			

В этом примере, кстати, Т_ЭХ никогда не разорвет строку между словами «осел» и «козел», — и никогда не сделает переносов в этих словах: при верстке абзаца Т_ЭХ 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 Т_ЭХ не сможет растянуть или сжать пробел между словами «осел» и «козел» для выравнивания строк в абзаце.

Теперь, когда мы знаем, что такое Т_ЭХовские блоки, можно признать, что окружения `picture`, `array` и `tabular` тоже генерируют блоки, и именно поэтому создаваемый ими текст воспринимается Т_ЭХом как одна большая буква.

В аргументе команды `\mbox` может присутствовать все то же, что может быть в обычном тексте в пределах одной строки: математические формулы, команды смены шрифта или присваивания значений каким-то параметрам, команды для генерации блоков (например, тот же `\mbox`, или даже окружения `picture` или `array`), и т. д. Запрещены в аргументе команды `\mbox` пустые строки или команды `\par`, выключные математические формулы, окружения, определяющие абзацы специального вида (наподобие `itemize` или `center`), команда `\` и тому подобные вещи, «не вписывающиеся в строку». Если в аргументе `\mbox` присутствуют команды смены шрифта, изменения каких-то параметров или определения 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 <code>\makebox[5em]{и}</code>	обратно.
------	---	----------	------------------------------------	----------

Как видите, необязательный аргумент ставится перед обязательным; длина в нем может быть указана, как обычно, либо в какой-либо из Т_ЕХовских единиц, либо через какой-либо параметр со значением длины, возможно — с числовым коэффициентом (см. раздел VII.3). 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 будет вылезать за края блока; поскольку место, отводимое Т_ЕХом блоку при верстке, 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_ЕХа, так (для ясности мы использовали в этом примере крупный шрифт):

123456

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. Для этого в команде `\makebox` предусмотрен второй необязательный аргумент — буква `l` для текста, прижатого влево или `r` для текста, прижатого 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. Кстати, обратите внимание, что у нас получилась верстка с выравниванием без помощи таких вещей, как `tabbing` или `tabular`.

У команды `\makebox` значение ширины блока можно установить равным нулю. Если при этом присутствует необязательный аргумент `l`, то получится блок нулевой ширины, а текст будет выходить за его пределы вправо (и, стало быть, наложится на последующий текст в строке, если таковой присутствует); если присутствует необязательный аргумент `r`, то текст будет выходить влево за пределы блока (и тем самым накладываться на предшествующий текст):

тексттекст	текст\makebox[0pt][l]{???}текст\
тексттекст	текст\makebox[0pt][r]{???}текст

2.2. Блоки из абзацев

Если необходимо создать блок, в котором размещается сверстанный Т_ЭХом 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 Т_ЭХовским прерванная строка.
 правилам. После это-
 го продолжается

получился следующим образом:

В строку `\quad`
`\parbox{4cm}{вставили целый`
 абзац текста, сверстанного
 по всем `\TeX` овским правилам.
 После этого продолжается `\quad`
 прерванная строка.

Как видите, базисная линия блока, создаваемого командой `\parbox`, находится в точности посередине текста. По этой причине 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 Ри-
 ману.

получается из такого исходного текста:

```
$$
\int_a^b f'(x) dx = f(b) - f(a) \quad
\parbox{4cm}{для всех функций $f$,
производная которых интегрируема
по Риману.}
$$
```

Если дать команду `\parbox` с необязательным аргументом, то создаваемый ею блок можно расположить относительно строки и по-иному: чтобы вровень с остальной строкой шла самая верхняя строка абзаца (для этого нужен аргумент `t`) или самая нижняя (аргумент `b`) (можно также указать аргумент `c` — тогда блок будет расположен по центру, так же, как если бы необязательного аргумента вообще не было. Необязательный аргумент у этой команды должен идти перед обязательными).

Во втором обязательном аргументе команды `\parbox`, задающем текст, может присутствовать всё то же, что в обычном тексте, в том числе команды для пробелов по вертикали наподобие `\vspace`, пустые строки, разделяющие абзацы, выключные формулы и т. п. Абзацы, создаваемые командой `\parbox`, по умолчанию делаются без абзацного отступа и в режиме `\slippy`. Если Вы хотите чего-то другого, можно прямо внутри аргумента команды `\parbox` установить нужное Вам значение абзацного отступа, параметра `\tolerance` и т. п. (см. раздел III.6 по поводу смысла этих параметров).

Наряду с `\parbox`, существует еще один, довольно экзотический, способ создать блок из абзацев. Именно, существует окружение `minipage` («министраница»), генерирующее блок из текста, расположенного внутри этого окружения; блок состоит из абзацев, ширина которых задается в обязательном аргументе окружения `minipage` (так же, как в команде `\parbox`); перед обязательным аргументом этого окружения может стоять необязательный: буква `t`, `b` или `c`, причем смысл этого аргумента опять-таки такой же, как в команде `\parbox`. Основное отличие `minipage` от `\parbox` в том, что к тексту внутри этого окружения можно делать сноски с помощью команды `\footnote`, причем текст сноски появляется *не* внизу страницы, а внизу блока, генерируемого окружением `minipage`. При верстке книги, которую Вы читаете, это окружение использовалось для печати примеров; придумать разумное применение этой экзотической конструкции вне учебных пособий по ЛАТЭХу автору не удалось.

2.3. Текст в рамке; комбинации блоков

В главе 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` соответственно). Смысл и расположение обязательных и необязательных аргументов у команды `\framebox` такой же, как и уых 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, равным значению параметра `\fboxsep`; толщина линий в рамке равна значению параметра `\fboxrule`. Обоим этим параметрам можно обычным образом присваивать новые значения (см. раздел VII.3).

Коль скоро каждый блок, создаваемый ЛАТЭХовскими командами, рассматривается ТЭХом просто как большая буква, возможны любые, сколь угодно причудливые, комбинации таких «букв». Пусть, например, нам надо взять в рамку абзац текста шириной 6 см, чтобы получилось так:

Внутри ТЭХовских блоков может присутствовать не только собственно текст или формулы, но и другие блоки, внутри этих блоков — еще блоки, и так далее. Таким образом, блоки могут быть вложены друг в друга, как матрешки.

Просто поместить этот текст в аргумент команды `\fbox` не получится, поскольку наш текст в одну строку не укладывается, а команда `\fbox`, подобно команде `\mbox`, текстов, не укладывающихся в строку, не переваривает. Поэтому нужно сделать из нашего абзаца блок с помощью команды `\parbox` и этот блок (то есть уже «букву») передать в качестве аргумента команде `\fbox`:

```
\fbox{%
\parbox{6cm}{%
Внутри \TeX{}овских блоков может ...
... друг в друга, как матрешки.}%
}
```

Обратите внимание на знаки процента, которыми заканчиваются первая и предпоследняя строка. Если бы их не было, то рамка отстояла бы от текста больше, чем надо, так как Т_ЭХ решил бы, что аргумент команды `\fbox` имеет пробел до и после «буквы», созданной командой `\parbox`. См. стр. 11 по поводу использования знака процента для удаления нежелательных пробелов.

2.4. Сдвиги относительно базисной линии

Когда при исполнении команды `\makebox` или `\mbox` Т_ЭХ создает блок из меньших блоков (каждая буква, как мы помним, — это блок, из букв составляются слова — тоже блоки, и, наконец, блоки могут быть заданы в явном виде, в частности, командой `\mbox`), то блоки эти размещаются в строке таким образом, что все их точки отсчета расположены на одной высоте (иными словами, их базисные линии продолжают одна другую). Можно, однако, сдвинуть блок по вертикали относительно базисной линии. Для этого удобно воспользоваться ЛАТЭХовской командой `\raisebox`. Эта команда требует двух обязательных аргументов. Первый из них — расстояние, на которое сдвигается по вертикали фрагмент текста, второй — сам этот фрагмент текста. Пример:

Слово подскочило в строке.	Слово <code>\raisebox{2pt}{подскочило}</code> в строке.
----------------------------	---

Текст, расположенный во втором обязательном аргументе этой команды, должен удовлетворять тем же требованиям, что и аргумент команды `\mbox`: в нем могут быть самые разные Т_ЭХовские команды, при условии, что среди них не будет команд типа пустой строки, `\par`, `\` и тому подобных, которые «не лезут в строку» (зато в этом тексте, как водится, могут присутствовать любые команды, порождающие блоки, в частности, например, `\parbox`, а уж в ее аргументе оставляйте пустых строк, сколько душе угодно). Если первый обязательный аргумент команды `\raisebox` отрицателен, то текст будет, естественно, не поднят, а опущен. Вот, например, как можно определить команду `\TeX`, печатающую эмблему Т_ЭХа:

```
\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 сблизилась. Отметим, что команды `\nolinebreak` использованы, 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 Т_ЭХа. Время от времени мы будем приводить определения команд «в переводе с Т_ЭХа на ЛАТЭХ», в виде, более понятном читателям этой книги.

Кроме вертикального сдвига блоков, команда `\raisebox` может делать еще одно полезное дело: с ее помощью можно обмануть Т_ЭХ, 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 Т_ЭХа, должен иметь сдвинутый блок. Кроме того, после первого необязательного аргумента может стоять второй — глубина, которую, по мнению Т_ЭХа, будет иметь сдвинутый блок. Вот пример:

Строчка.	Строчка. \\
Вторая	Вторая
Вторая	<code>\raisebox{7pt}[1pt][10pt]{Ы} \\</code>
Третья строчка.	Третья строчка.

Буква Ы, поднятая на 7 пунктов над строчкой, наложилась на первую строку, поскольку в первом необязательном аргументе команды `\raisebox` мы приказали Т_ЕХу считать, что блок, образованный поднятой буквой Ы, имеет высоту всего лишь один пункт (стало быть, возвышается над базисной линией второй строчки меньше, чем любая буква), и соответственно Т_ЕХ не сделал дополнительного интервала между первой и второй строками. С другой стороны, третья строка отъехала от второй, поскольку во втором необязательном аргументе команды `\raisebox` мы велели Т_ЕХу считать, что глубина блока, образованного поднятой буквой Ы, равна аж десяти пунктам, и Т_ЕХ послушно оставил дополнительное место, чтобы этот блок не наложился на третью строчку!

Иногда разумно использовать команду `\raisebox` даже с нулевым обязательным аргументом, только для того, чтобы менять (в глазах Т_ЕХа) высоту и/или глубину блока, не сдвигая его относительно базовой линии. В главе IX мы увидим пример такого использования этой команды.

3. Команда `\hbox`

Возможности, предоставляемые Л^AT_EXом для генерации блоков, достаточны для простых приложений, но в более серьезных случаях их не хватает. В этом и следующем разделах мы рассмотрим более гибкие средства, предоставляемые для этой цели непосредственно языком Т_ЕХ и макропакетом Plain Т_ЕХ. Мы не будем пытаться описать все Т_ЕХовские команды для генерации блоков (книгу [2] ничто заменить не может), но сообщим тот необходимый минимум сведений, который необходим для модификации Л^AT_EXовских стандартных стилей, о чем пойдет речь в следующей главе. Подчеркнем, что всеми описываемыми в этом и следующем разделах Т_ЕХовскими средствами можно пользоваться в Л^AT_EXовских исходных текстах.

Прежде всего давайте вспомним (стр. 78), что в каждый момент трансляции исходного текста Т_ЕХ 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 Л^AT_EXовской команды для генерации блока или линейки (к таковым относятся `\mbox`, `\makebox`, `\fbox`, `\framebox`, окружения `array` или `picture`, а также команда `\rule`¹) Т_ЕХ из вертикального режима выходит и начинает очередной абзац.

Одна из основных Т_ЕХовских команд для генерации блоков называется `\hbox`. В своем простейшем виде она полностью аналогична Л^AT_EXовской команде `\mbox`, с одним важным отличием: в вертикальном режиме команда `\hbox` *не* начинает нового абзаца, а только добавляет сгенерированный ею блок (то есть фактически строчку) к уже сверстанной части страницы. Внутри абзаца (по-ученому: в горизонтальном режиме) команда `\hbox` действует точно так же, как и `\mbox`. Вот пример:

¹Но не `\hrule` или `\vrule`: это команды Т_ЕХовские, а не Л^AT_EXовские.

На странице уже присутствует абзац текста. После того, как он кончится, `TeX` перейдет в вертикальный режим.

Строчка

Еще строчка

Только теперь начинается новый абзац.

На странице `\hbox{уже}` присутствует абзац текста.

После того, как он кончится, `\TeX{}` перейдет в вертикальный режим.

`\hbox{Строчка} \hbox{Еще строчка}` Только теперь начинается новый абзац.

Сравните с тем, что было бы при использовании `LaTeX`овской команды `\mbox` вместо `\hbox`:

На странице уже присутствует абзац текста. После того, как он кончится, `TeX` перейдет в вертикальный режим.

Эти слова сразу начинают новый абзац.

На странице уже присутствует абзац текста. После того, как он кончится, `\TeX{}` перейдет в вертикальный режим.

`\mbox{Эти слова}` сразу начинают новый абзац.

3.1. Растяжимые интервалы

До сих пор шла речь о важных, но принципиальных отличиях между `TeX`овским `\hbox`'ом и `LaTeX`овским `\mbox`'ом. Теперь поговорим о дополнительных возможностях, предоставляемых `TeX`овской командой.

Команда `\hbox` «в чистом виде» создает блок, ширина которого равна естественной длине текста, являющегося ее аргументом. Кроме этого, она может создавать блоки, любой заданной ширины. Для этого нужно сказать

`\hbox to <ширина> {текст}`

Здесь `<ширина>` должна быть выражена в воспринимаемых `TeX`ом единицах длины: это может быть, например, `20pt`, или `2.3cm`, или, например, `0.12\textwidth` — параметр со значением длины (возможно, с коэффициентом) тоже годится. Между `to` и обозначением ширины, а также между обозначением ширины и открывающей фигурной скобкой могут быть пробелы — `TeX` их проигнорирует.² Наконец, отсутствие `backslash`'а в слове `to` не является опечаткой: это не команда, а одно из «ключевых слов» `TeX`а (подобно ключевым словам `plus` и `minus`, с которыми мы вскоре снова встретимся, или `width` и `height`, с которыми мы уже встречались в разделе, посвященном линейкам). Давайте опробуем эту новую возможность команды `\hbox`:

Два слова

`\hbox to 3cm {Два слова}`

Если Вы действительно опробовали этот пример на Вашем компьютере, то заметили, что на экране появилось сообщение

Underfull \hbox

²Пустых 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'e`, как это было объяснено в разделе III.6.5Б.

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` присутствуют несколько `\hfil`'ов, то растяжение произойдет на месте каждой из этих команд, причем размер этого растяжения будет распределен между `\hfil`'ами 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 два `\hfil`'а стоят по обе стороны от текста, то текст внутри блока будет центрирован, поскольку дополнительное растяжение поделится между двумя `\hfil`'ами поровну:

Слева	<code>\hbox to 0.7\textwidth {Слева\hfil}</code>
Справа	<code>\hbox to 0.7\textwidth {\hfil Справа}</code>
В центре	<code>\hbox to 0.7\textwidth {\hfil В центре\hfil}</code>

Можно считать, что на месте каждого `\hfil`'а 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`'ом. Если в аргументе команды `\hbox` присутствуют `\hfil` и `\hfill` совместно, то все растяжения происходят только за счет «более растяжимых» `\hfill`'ов:

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. Лидеры

В оглавлении к этой книге (и ко многим другим тоже) место между названием раздела и номером страницы заполняется рядом из точек. T_EX дает возможность печатать ряды из точек, заполняющие заданный пробел. Для этой цели служит L^AT_EXовская команда `\dotfill`. Она работает так же, как и `\hfill`, с той разницей, что пробел, образующийся в результате действия этой команды, заполняется точками:

А Б `\hbox to 3cm{A\dotfill B}`

Кроме этого, есть L^AT_EXовская команда `\hrulefill`, которая также действует аналогично команде `\hfill` и при этом заполняет пробел линейкой:

1 _____ 2 _____ 3 `\hbox to 5cm{1\hrulefill`
`2\hrulefill 3}`

В T_EXнической терминологии такие заполнители называют лидерами (leaders по-английски).

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 ЪЪЪЪЪЪЪЪЪЪЪЪ 2 `\hbox to 5cm{1\leaders`
`\hbox{Ъ}\hfil 2}`

Если бы мы хотели, чтоб буквы Ъ шли не вплотную, можно было бы, например, вместо `\hbox{Ъ}` написать так:

`\hbox to 2em{\hfil Ъ\hfil}`

В общем случае применяйте команду `\leaders` так:

`\leaders < блок > < \hfil или \hfill >`

Здесь `< блок >` — это любая T_EXовская команда для генерации блока, например, `\hbox`, с которой мы уже познакомились, или `\vbox` или `\copy`, о которых еще пойдет речь. L^AT_EXовские команды `\mbox`, `\makebox`, `\parbox` и тому подобные применять в этом месте 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). Работает команда `\leaders` 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 >`, а, 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 задать по своему усмотрению 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 `\hfил` вместо `\hfill`, с очевидными последствиями.

3.3. Клей

Выше мы рассмотрели команды `\hfил` и `\hfill`, которые действуют подобно вставленным в строку пружинам. Т_ЭX позволяет вставлять в строку пружины с самыми разнообразными свойствами. Для этого применяется команда `\hspace`, аргумент которой содержит `plus`- или `minus`-компоненту. Мы упоминали об этой возможности в разделе III.7.3, но в тот момент у нас еще не было серьезных примеров. Теперь давайте займемся такими командами вплотную. 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 максимум на y ³ и уменьшать свою длину максимум на z (в отличие от пружин, встречающихся в жизни, может выполняться неравенство $x < z$, и, того хуже, длины y и z могут быть отрицательными, но мы не будем объяснять, как Т_ЭX поведет себя в столь странной ситуации). Здесь `plus` и `minus` — это, как мы помним, очередные ключевые слова Т_ЭXа, 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_ЭX создать блок, ширина которого отличается от естественной, то для достижения требуемой ширины размеры пробелов будут изменяться. В Т_ЭXнической терминологии эти «пружины» называются *клеем* (Дональд Кнут отмечает, что название «клей» неудачно, но менять его поздно, поскольку, по его словам, «оно уже приклеилось»). 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 `\hfил` 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 `\hfил`'ов или `\hfill`'ов; про клей с бесконечной сжимаемостью речь пойдет ниже). В этом случае Т_ЭX 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_ЭX вычисляет, насколько больше, после чего «разверстывает» эту добавку между всеми пробелами пропорционально величинам `plus`-компонент клея в этих пробелах. Вот пример. Предположим, мы создаем блок с помощью команды

³Если мы заставим ее растянуться больше, чем на y , то получим сообщение «Underfull `\hbox`»; см. ниже.

```
\hbox to a {\hspace{0pt plus 2em}%
  Б\hspace{1cm plus 1em minus 2mm}В}
```

где величина a на три миллиметра больше естественной ширины блока АБВ, то пробел между А и Б будет равен двум миллиметрам, а пробел между Б и В — одиннадцати миллиметрам, поскольку `plus`-компонента клея между А и Б в два раза больше, чем `plus`-компонента клея между Б и В (и никакого другого клея в строке нет, так что ничего более растянуть нельзя). Если требуемая ширина меньше естественной, то уменьшение длины также разверстывается 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 вышеприведенном 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 стр. 76, 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 хорошо 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, 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 `\hfil`'ов. 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.

3.4. Бесконечно сжимаемые интервалы

Мы уже два раза упомянули про клей с бесконечной сжимаемостью. Настало время объяснить, какими командами его создавать. Из многих способов укажем один, наиболее часто встречающийся. Команда `\hss` вставляет в строку клей, естественный размер которого равен нулю, и который при этом обладает бесконечной растяжимостью (подобно `\hfil`) и бесконечной сжимаемостью. Типичное применение такого «бесконечно сжимаемого» клея — создавать блоки, ширина которых меньше реального размера текста, или блоки с наложением текстов. В самом деле, посмотрите на такой пример:

Кошка	Собака	<code>\hbox to 4cm {Кошка\hss Собака}</code>
КошкаСобака		<code>\hbox{Кошка\hss Собака}</code>
К о Шка		<code>\hbox to 17mm {Кошка\hss Собака}</code>
Соба к а		<code>\hbox to 5mm {Кошка\hss Собака}</code>
СобакаКошка		<code>\hbox to 0mm {Кошка\hss Собака}</code>

Когда мы просим, чтобы ширина блока превышала естественную, команда `\hss` действует так же, как и `\hfil`; когда мы создаем блок с естественной шириной, слова «Кошка» и «Со-

бака» стоят вплотную друг к другу (естественная ширина клея, созданного `\hss`, равна нулю). Интересные вещи начинаются, когда мы просим, чтобы ширина была 17 мм (что меньше естественной). Интервал между словами при этом приходится уменьшить; поскольку его естественный размер равен нулю, то после уменьшения интервал становится отрицательным, то есть слово «собака» сдвигается влево (накладываясь на слово «Кошка»), причем сдвигается так, чтобы ширина блока (то есть расстояние от начала слова «Кошка» до конца слова «Собака») равнялась требуемым 17 мм. Когда же мы наконец просим, чтобы ширина блока равнялась нулю, слову «Собака» приходится сдвинуться влево настолько, чтобы расстояние от его конца до начала слова «Кошка» равнялось нулю — иными словами, кошка и собака меняются местами! Заметим, кстати, что точка отсчета всех наших блоков совпадает с точкой отсчета буквы К из слова «Кошка».

Еще один пример использования `\hss`: как создать блок, точка отсчета которого будет находиться в правом, а не левом конце текста (мы столкнулись с этой проблемой в главе V — см. стр. 116)? Ответ: надо сказать

```
\hbox to 0pt{\hss текст}
```

и все будет в порядке. В самом деле, *текст* имеет ширину, отличную от нуля; чтобы блок имел в итоге нулевую ширину, приходится «уменьшать» тот интервал, где стоит `\hss`; так как интервал уже нулевой, то это уменьшение сводится к тому, что текст сдвигается влево, до тех пор, пока расстояние между его концом и точкой отсчета не станет равным нулю — а это и означает, что правый конец текста стал его точкой отсчета. В главе V мы сказали, что эта проблема решается с помощью Т_ЕХовской команды `\llap`, а теперь мы видим, как ее можно определить:

```
\newcommand{\llap}[1]{\hbox to 0pt{\hss #1}}
```

Кстати говоря, именно так она и определяется.

А если сказать

```
\hbox to 0pt{текст\hss}
```

то что, спрашивается, будет? Ответ: на сей раз будет уменьшаться интервал *после* текста; стало быть, сам текст никуда не сдвинется, но после него будет сделан такой «отрицательный пробел», чтобы суммарная ширина равнялась нулю. Иными словами, Т_ЕХ будет просто считать, что ширина блока равняется нулю — мы обманули Т_ЕХ, убедив его, что наш текст не занимает места по горизонтали! Для такого обмана (к нему приходится прибегать нередко) предусмотрена специальная Т_ЕХовская команда `\rlap`, определяемая так:

```
\newcommand{\rlap}[1]{\hbox to 0pt{#1\hss}}
```

Все это также напоминает ситуацию с командой `\lefteqn`, и напоминает не случайно, поскольку эта команда определяется фактически так:

```
\newcommand{\lefteqn}[1]{\rlap{\$ \displaystyle
#1 \$ \hss}}
```

3.5. Еще раз о линейках

В аргументе команды \hbox может присутствовать и Т_ЕXовская 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 разделе 9 главы III, можно задать в явном виде ширину линейки с помощью ключевого слова `width`, высоту — с помощью ключевого слова `height`, а также (о чем в главе III не говорилось) глубину с помощью ключевого слова `depth` (эти три ключевых слова могут следовать после \vrule в произвольном порядке). Приведем один неочевидный 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:

```
\noindent\hbox{%
\vrule\hspace{.5em}\parbox{.9\textwidth}%
{Иногда используется ...
... линейка.}}
```

Этот текст, надо думать, нуждается в некоторых пояснениях. Во-первых, в последней строчке первая из фигурных скобок закрывает аргумент команды \parbox, а вторая — \hbox. Во-вторых, текст начинается с команды \noindent. Мы предполагаем, что предыдущий абзац уже завершен, и теперь хотим, чтоб наш огромный \hbox начал абзац без отступа (в самом тексте — аргументе команды \parbox абзацный отступ будет, как всегда в блоках, созданных командой \parbox, нулевым; при желании мы могли бы начать аргумент команды \parbox с присваивания нового значения параметру \parindent). Параметр \textwidth означает, как мы помним, ширину страницы. Теперь рассмотрим, что же присутствует в аргументе \hbox'а? Сначала линейка, затем тонкий пробел, созданный командой \, и поле него — 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'е нет). Тем самым линейка получается как раз нужных размеров, что и требовалось!

Обратите еще внимание на знак процента, завершающий первую строку. Без этого знака получилось бы, что аргумент \hbox'а начинается с пробела, соответственно и линейка началась бы не с начала, а после пробела (ср. стр. 11).

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 \ширина и \разность, сделаем вот что:

```
\ширина=\textwidth
\settowidth{\разность}{\vrule\hspace{.5em}}
\addtolength{\ширина}{-\разность}
\noindent\hbox{%
\vrule\hspace{.5em}\parbox{\ширина}%
{Иногда используется ...
... линейка.}}
```

Мы воспользовались командой `\settowidth`, чтобы найти размер, который занимает линейка вместе с тонким пробелом. Кстати, если 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 0,4 пункта плюс размер тонкого пробела.

4. Команда `\vbox`

Теперь рассмотрим вторую основную Т_ЭХовскую команду 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:

Слово	⇒	Слово
Еще слово		Еще слово

Как видите, блоки, создаваемые `\hbox`'ами, 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 Т_ЭХовские команды, допустимые между абзацами (то есть 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 мы будем считать, что из них в аргументе `\vbox` возможны только Т_ЭХовские команды `\hbox`, `\vbox` и `\hrule`, а также `\copy`, о которой пойдет речь позже. В частности, недопустим ни текст, ни Л^AТ_ЭХовские 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 Л^AТ_ЭХовской команды для генерации блоков Т_ЭХ переходит в горизонтальный режим, который продолжается до команды, завершающей абзац (`\par` или пустой строки). Мы не будем вдаваться в детали; для тех приложений, которые мы имеем в виду, достаточно использовать команду `\vbox` так, как было предписано выше.

Когда Т_ЭХ 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 дружки, 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 рамочку:

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.

5. Блочные переменные

В тех случаях, когда один и тот же фрагмент текста (например, фрагмент псевдорисунка, являющийся аргументом команды `\multiput`) используется многократно, бывает полезно сверстать этот фрагмент раз и навсегда, а затем просто повторять его: это сэкономит как количество нажатий на клавиши, так и машинное время. Использование макроопределения в данном случае времени не сэкономит: если мы напишем, например,

```
\newcommand{\abcd}{\parbox{6cm}%
{Когда в товарищах согласья нет,
на лад их дело не пойдет, и выйдет
из него не дело --- только мука.}}
```

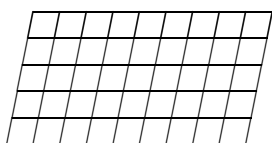
то при каждой обработке команды `\abcd` это макроопределение будет заново разворачиваться, и `TeX` будет заново находить переносы и места разрыва строк в одном и том же отрывке из «Лебедя, рака и щуки». Чтобы не заставлять `TeX` много раз повторять идентичные операции по верстке текста, надо сделать так. Во-первых, определим «блочную переменную», которая будет хранить сверстаный фрагмент текста. Это делается с помощью команды `\newsavebox`. Единственный аргумент этой команды — имя новой блочной переменной, которое должно удовлетворять тем же условиям, что любые имена `TeX`овских команд: либо `backslash` с одной не-буквой, либо `backslash` с последовательностью букв. Имя новой блочной переменной не должно совпадать с именем уже существующей команды или переменной длины (если Вы попытаетесь нарушить это правило, `LaTeX` выдаст сообщение об ошибке). Во-вторых, присвоим нашей блочной переменной значение — блок, и будем в дальнейшем этот блок использовать. Давайте приведем пример того, как можно этим пользоваться. На стр. 119 мы приводили пример псевдорисунка-наклонной решетки, и там же мы отметили, что экономнее было бы сверстать наклонный отрезок раз и навсегда, а затем только повторять его. Теперь мы можем объяснить, как это сделать. Создадим блочную переменную под названием `\блок`, написав в преамбуле следующее:


```
\newsavebox{\блок}
```

Теперь сверстаем тот текст, который будет храниться в нашей блоковой переменной, и запишем его в эту переменную. Для этих целей используется команда `\sbox` с двумя обязательными аргументами: первый — имя блоковой переменной, второй — текст, который в нее записывается. Итак:

```
\sbox{\блок}{\line(1,5){10}}
```

А теперь можно воспользоваться нашей блоковой переменной. Чтобы напечатать содержимое блоковой переменной, используется команда `\usebox` с одним обязательным аргументом — именем переменной. В нашем случае мы используем блоковую переменную в аргументе команды `\multiput`:



```
\begin{picture}(100,50)
\multiput(0,0)(10,0){10}%
{\usebox{\блок}}
\multiput(0,0)(2,10){6}%
{\line(1,0){90}}
\end{picture}
```

Можно было бы сделать аналогичный трюк и с горизонтальными линиями решетки, но большой экономии это не даст: горизонтальные и вертикальные линии на псевдорисунках \LaTeX не собирает из отдельных символов, а создает «в один присест» с помощью команды `\vrule`, что и так достаточно быстро.

Текст, присутствующий в аргументе команды `\sbox`, будет сверстан в виде блока так, как если бы этот текст был передан в качестве аргумента команде `\hbox` или `\mbox`. Тем самым в аргументе `\sbox` может быть все то же, что может присутствовать в аргументе `\hbox` или `\mbox`. Если команда `\sbox` была дана внутри группы, то по выходе из этой группы содержимое блоковой переменной забудется.

Наряду с командой `\sbox` есть еще и команда `\savebox`, относящаяся к ней примерно так же, как `\makebox` относится к `\mbox`: между первым и вторым обязательным аргументом команды `\savebox` могут присутствовать необязательные аргументы, имеющие тот же смысл и записывающиеся так же, как необязательные аргументы команды `\makebox`. Например, написать

```
\savebox{\пример}[4cm][r]{Слово}
```

все равно, что

```
\sbox{\пример}{\makebox[4cm][r]{Слово}}
```

Наряду с \LaTeX овской командой `\usebox` есть похожая на нее, но не идентичная, \TeX овская команда `\copy`. Используется она так:

Однажды Лебедь, Рак и Щу-
ка...

```
\sbox{\блок}{Рак}
\copy\блок{ и Щука\ldots}
```

Обратите внимание, что при использовании команды `\copy` имя блоковой переменной *не* заключается в фигурные скобки! Различие между `\copy` и `\usebox` — такое же, как между `\hbox` и `\mbox`: будучи употребленными внутри абзаца (или, скажем, в аргументе команд `\put`, `\hbox` или `\mbox`) эти две команды действуют совершенно одинаково, а вот будучи употребленным между абзацами, Л^AT_EXовское `\usebox` начинает новый абзац, в то время как Т_EXовское `\copy` просто подверстывает блок к странице, нового абзаца не начиная. Эту разницу следует иметь в виду, когда Вы работаете с командой `\leaders`: выгоднее сверстать блок один раз и записать его в блоковую переменную, а затем в команде `\leaders` писать просто `\copy`. Пример:

```
* * * * * \savebox{\блок}[1cm]{*$*}$
\hbox to \textwidth
{\leaders\copy\блок\hfil}
```

В этой ситуации по Т_EXническим причинам сказать `\usebox` нельзя.

Глава IX.

Модификация стандартных стилей

Эта глава предназначена для тех, кого не удовлетворяет оформление, навязываемое нам стандартными стилями Л^AT_EX. Возможно, Вам даже захочется создать свой собственный стиль вместо стандартных `article`, `report` или `book`. Задача эта выполнимая, но для этого надо в деталях знать во-первых, книгу [2], а во-вторых — исходные тексты Л^AT_EX (они доступны). У читателя настоящей книги таких познаний не предполагается, так что мы предлагаем нечто более скромное: создание собственной *стилевой опции*, позволяющей модифицировать оформление, задаваемое одним из стандартных Л^AT_EXовских стилей.

Прежде, чем двигаться дальше, — два предупреждения. В этой главе мы расскажем Вам, как можно весьма сильно изменить оформление документа, задаваемое стандартными Л^AT_EXовскими стилями: Вы научитесь менять по своему усмотрению шрифты в заголовках, интервалы, отделяющие заголовки от текста, и много других подобных вещей. Если Вы не являетесь профессиональным полиграфистом, применяйте эти познания с осторожностью. Не пытайтесь изменять сразу много разных черт оформления или резко изменять какие-то параметры оформления: лучше осторожно менять только то, что Вам действительно нужно. Учтите, что когда неспециалист берется за оформление книги, то в девяти случаях из десяти результат бывает достоин лишь сожаления.

Второе предупреждение относится вот к чему. Стиль оформления записан в специальных «стилевых файлах», входящих в комплект поставки Л^AT_EX. НИ В КОЕМ СЛУЧАЕ НЕ МЕНЯЙТЕ НИЧЕГО В ЭТИХ ФАЙЛАХ: все изменения в стиле надо записывать в отдельный, Ваш личный стилиевой файл, как это объяснено ниже.

1. Еще раз о стилях и стилиевых опциях

Кое-какие изменения в оформлении документа Вы делать уже умеете: например, в главе IV рассказывалось, как можно, присвоив в преамбуле новые значения нескольким параметрам, изменить размер полей или текста. Однако же для более серьезных дел команд, которые можно поместить в преамбуле, не хватает. Поэтому начать придется с рассказа о том, что конкретно представляют собой стили и стилиевые опции.

Стиль документа задается командой `\documentstyle`. Обязательный аргумент этой команды представляет собой имя так называемого «стилевого файла» с расширением `.sty`: стилю `article` соответствует файл `article.sty`, стилю `report` — файл `report.sty`, и т. д. Стилиевой файл содержит Т_EXовские команды, которые присваивают значения различным параметрам и задают различные макроопределения (например, в файле `book.sty` содержится определение команды `\chaptername`, ответственной за то, что главы называются Chapter).

Действие команды `\documentstyle` начинается с того, что \TeX считывает содержимое этого файла; при этом происходят все вышеупомянутые присваивания и определения макросов.

После того, как основной стилевой файл прочитан, начинается просмотр списка стилевых опций, который, как Вы помните, задается в качестве необязательного аргумента команды `\documentstyle`. Если в этом списке присутствует стилевая опция, название которой отличается от стандартных, то \TeX читает «дополнительный стилевой файл» с именем, совпадающим с именем опции, и расширением `.sty`: если, например, Вы сказали

```
\documentstyle[abcd]{article}
```

то \TeX будет читать файл `abcd.sty`, а если такового не найдет, то выдаст Вам ошибку

```
! I can't find file 'abcd.sty'.
```

Чтобы модифицировать стандартный стиль, надо создать дополнительный стилевой файл и включить в необязательный аргумент команды `\documentstyle` его имя. Что именно писать в этот файл, мы расскажем далее в этой главе.

Имя Вашего стилевого файла должно стоять в списке стилевых опций последним (в противном случае стилевые опции, исполняющиеся после Вашей, могут заново изменить установленные Вами значения параметров). Если Вы создаете собственный стилевой файл, то в него можно включить и все те команды, которые Вы записали бы в преамбулу документа (после этого повторять их в преамбуле, естественно, не надо).

Разумеется, стилевые файлы нужны не для того, чтобы записывать в них то же, что можно написать и в преамбуле. Важнейшая особенность \LaTeX овских стилевых файлов — то, что в них используются команды, содержащие в своем имени символ `@`. При обработке \LaTeX овских стилевых файлов \TeX рассматривает этот символ как букву, и тем самым он допускается в (неодносимвольных) именах команд (см. раздел 1.2.3). Для модификации стилей часто приходится переопределять команды или присваивать значения параметрам, в имени которых присутствует `@`, и именно поэтому мы рассказываем, что такое стилевой файл.

Если изменения параметров и/или переопределения команд присутствуют как в стилевых файлах, так и в преамбуле документа, то имейте в виду, что присваивания и переопределения, заданные в преамбуле, выполняются *после* тех, что заданы в стилевом файле. Если Вы задали несколько собственных файлов-стилевых опций, то записанные в них команды исполняются в том же порядке, в каком ссылки на эти файлы записаны в необязательный аргумент команды `\documentstyle`.

После всех этих предупреждений пора приступать к делу.

2. Снова о счетчиках

Для начала расскажем об еще двух манипуляциях со счетчиками, которые иногда бывают полезны. До сих пор обходили их молчанием, поскольку команды, используемые для этих манипуляций, содержат `@` в своих именах и могут поэтому использоваться только в стилевых файлах.

Первый из приемов, о которых пойдет речь, относится к отношению подчинения между счетчиками. Мы знаем, что при создании счетчика с помощью команды `\newcounter` можно задать и счетчик, которому он будет «подчинен» (см. стр. 147 и ниже). Но как быть, если уже создан 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, из-за чего нумерация сносок выходит сплошной в пределах всего документа. Если мы хотим, чтоб нумерация сносок начиналась заново в каждой секции, то можно в своем стилевом файле написать так:

```
\@addtoreset{footnote}{section}
```

Первый аргумент команды `\@addtoreset` — имя подчиняемого счетчика, второй — имя подчиняющего.

Разумеется, для того, чтобы осмысленно применять описанную команду, надо знать, какие счетчики определены в стандартных стилях и кому они подчинены (или не подчинены). Вся эта информация содержится в приложении Г.

При ознакомлении с командой `\@addtoreset` может возникнуть искушение написать

```
\@addtoreset{footnote}{page}
```

чтобы сноски нумеровались заново на каждой странице. К сожалению, по Трехтехническим 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 основной стиль — `article`. В этом случае часто бывает разумно написать в стилевом файле

```
\@addtoreset{equation}{section}
```

чтобы нумерация уравнений была не сплошной, как предусмотрено стандартом, а начиналась заново в каждом разделе. Разумеется, в этом случае надо будет и переопределить команду `\theequation`.

Вторая обещанная тонкость связана с автоматическими ссылками и `the`-командами. Предположим, что перед исполнением команды `\label{метка}` последним увеличению с помощью `\refstepcounter` подвергался счетчик `abcd`. В главе VII мы говорили, что при этом команда `\ref{метка}` представит на печати значение этого счетчика «в соответствии с командой `\theabcd`». Настало время сознаться, что это — всего лишь полуправда. На самом деле команда `\ref` напечатает перед `\theabcd` еще и так называемый «ссылочный префикс» счетчика. Содержимое ссылочного префикса к счетчику `abcd` записано в команде `\p@abcd` (буква `p` — латинская). В момент создания счетчика эта команда определяется как макроопределение с «пустым» замещающим текстом, так что у таких счетчиков, как `chapter` или `section` в стандартных стилях, ее следов не видно. Можно, однако, переопределить эту команду, чтобы ссылочный префикс реально печатался. Вот пример работы со ссылочным префиксом.

В стандартном стиле `book` вид номера главы и номера секции 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 секции определен как

```
\renewcommand{\thesection}{\thechapter.\arabic{section}}
```

Из-за этого номер третьей секции второй главы печатается в заголовке как 2.3. Предположим, мы хотим, чтобы номера секций в заголовках не содержали номера главы; тогда можно написать

```
\renewcommand{\thesection}{\arabic{section}}
```

но при этом возникнет другая неприятность. Автоматические ссылки на номер секции, генерируемые командой `\ref`, теперь дадут на печати одно и то же число 3 как для третьей секции второй главы, так и для третьей секции 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 третью секцию второй главы и 4.3 при ссылке на третью секцию четвертой главы: хотя `\thesection` в обоих случаях дает просто 3, но печатающийся перед ним `\p@section` обеспечивает печать номера главы и точки. Кстати говоря, именно такой прием применен в стилевом файле, использованном при верстке книги, которую Вы читаете.

3. Разделы документа

Здесь мы расскажем о том, как менять оформление разделов документа, предписываемое стандартными ЛАТЭХовскими стилями.

3.1. Что нумеровать и что включать в оглавление

ЛАТЭХ определяет, какие разделы документа нумеровать, а какие — нет, исходя из двух вещей: «уровня вложенности» раздела и значения счетчика `secnumdepth`. Раздел документа получает номер, если уровень его вложенности меньше или равен значения `secnumdepth` (разумеется, все сказанное относится к случаю, когда ЛАТЭХовская команда для раздела документа дана без звездочки — иначе нумерации не будет заведомо). Уровни вложенности в стандартных стилях определяются так:

Название раздела	Уровень
<code>\section</code>	1
<code>\subsection</code>	2
<code>\subsubsection</code>	3
<code>\paragraph</code>	4
<code>\subparagraph</code>	4

Стало быть, если мы хотим, чтобы самый мелкий из нумеруемых разделов был `\subsection`, то надо написать в своем стилевом файле (или в преамбуле документа) команду

```
\setcounter{secnumdepth}{2}
```

Ниже мы объясним, как можно менять и значения уровней вложенности.

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, если и только если уровень вложенности раздела меньше или равен значения этого счетчика (опять-таки, если раздел вводится командой со звездочкой, то в оглавлении он отражен не будет).

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 ЛАТЭХовских стилях.

Почти все команды для создания разделов документа определяются в ЛАТЭХовских стилевых файлах через команду `\@startsection`. Например, команда `\section` определяется (не буквально, а по существу) так:

```
\newcommand{\section}{\@startsection{section}{1}{Opt}%
{-3.5ex plus -1ex minus -.2ex}{2.3ex plus .2ex}%
{\Large\bf}}
```

Как видите, в этом определении у команды `\@startsection` указаны шесть аргументов, в которых закодированы различные параметры оформления раздела. Разберем последовательно, что эти аргументы означают.

Первый из аргументов (в нашем случае `section`) — это имя счетчика, используемого для нумерации соответствующих разделов. Сам счетчик при этом *не* создается, его надо создать отдельно с помощью команды `\newcounter`. Счетчики с именами `section`, `subsection`, `chapter` и т. д. создаются в стандартных ЛАТЭХовских стилях, так что об этом Вам беспокоиться незачем; если, однако, Вы захотите определить или переопределить команду для создания разделов, использующую новый счетчик (скажем, если Вы напишете

```
\newcommand{\раздел}{\@startsection{abcd}...
```

или что-то в этом роде), то Вам придется еще и создавать счетчик `abcd`. На самом деле имя, заданное в первом аргументе команды `\@startsection`, является не только именем счетчика, но и вообще тем «внутренним именем», под которым ЛАТЭХ будет знать определяемый нами тип разделов документа; это «внутреннее имя» будет, как мы увидим в дальнейшем, использоваться еще при создании колонтитулов и сборе материала для оглавления. Если Вы решили в своем стилевом файле использовать команду `\@startsection` с «нестандартным» первым аргументом (скажем, `abcd`), то Вам заодно придется определить счетчик с именем `abcd`, который будет отвечать за нумерацию Ваших разделов, а также команду `\l@abcd`, которая будет отвечать за сбор материала для оглавления (см. подробности в разделе 4) и команду `\abcdmark`, отвечающую за передачу информации для колонтитулов (см. раздел 6), иначе получите массу сообщений об ошибке. Рекомендуем Вам не пользоваться без особой надобности командой `\@startsection` с «нестандартным» первым аргументом.

Второй аргумент (в нашем случае `1`) — это тот самый «уровень вложенности» раздела, о котором шла речь выше. Следующие три аргумента отвечают за отступы и пробелы, оставляемые вокруг заголовка раздела. Именно, третий аргумент задает отступ заголовка от левого поля (в нашем случае этот отступ равен нулю). Следующий, четвертый аргумент заслуживает того, чтоб обсудить его подробнее.

Четвертый аргумент команды `\@startsection` (в нашем случае это `-3.5ex plus -1ex minus -.2ex`) задает, если говорить кратко, величину вертикального отступа, оставляемого перед заголовком. При этом `plus` и `minus` означают, как водится, что этот пробел обладает растяжимостью и сжимаемостью (см. стр. 80): после `plus` указано, насколько, самое большее, можно растянуть дополнительный интервал перед заголовком раздела, а после `minus` — насколько, самое большее, можно его ужать (растягивать или ужимать приходится, если

надо, чтобы все страницы имели одинаковую высоту: такой режим, как мы помним, автоматически задается стилевой опцией `twoside` или стилем `book`, или же его можно задать, написав в преамбуле или стилевом файле команду `\flushbottom`). При желании `plus`- и/или `minus`-компоненту можно и не указывать, тогда пробел перед заголовком потеряет способность растягиваться и/или сжиматься. Но почему же эти расстояния отрицательные? Ответ таков. Если расстояния, указанные в четвертом аргументе команды `\@startsection`, отрицательны, то при определении отступа перед заголовком \TeX отбрасывает знаки `-` перед ними и принимает за «естественную» величину отступа, а также его `plus`- и `minus`-компоненты получающиеся положительные расстояния. Если, однако, расстояния были заданы как отрицательные, то первый абзац нашего раздела будет печататься без абзацного отступа, как и оформляются разделы в стандартных \TeX овских стилях, а если задать эти расстояния сразу как положительные числа, то такого подавления абзацного отступа не происходит. Так что если бы мы захотели, чтоб абзацный отступ в первом абзаце секции не подавлялся, то надо было четвертый аргумент задать как `3.5ex plus 1ex minus .2ex`.

Пятый аргумент (в нашем случае `2.3ex plus .2ex`) задает величину вертикального отступа *после* заголовка раздела. Этот аргумент тоже 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 `\Large\bf`, 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, как можно изменить стандартное оформление. Пусть нам хочется, чтобы команда `\section` порождала раздел документа, оформленный таким образом:

- Перед номером раздела стоит слово «Тема».
- Слово «Тема», номер раздела и его заглавие печатаются шрифтом «сансериф» размера `\Large`.
- Абзацный отступ в первом абзаце раздела не подавляется.

В таком случае в своем стилевом файле надо будет написать так:

```
\renewcommand{\section}{\@startsection{section}{1}{0pt}{3.5ex
plus 1ex minus .2ex}{2.3ex plus .2ex}{\Large\sf Тема\}}
```

Несколько замечаний к этому тексту. Во-первых, мы воспользовались `\renewcommand`, поскольку команда `\section` ранее уже была определена в основных стилевых файлах. Во-вторых, обратите внимание на «backslash с пробелом» в последнем аргументе: если бы его не было, слово «Тема» печаталось бы вплотную к номеру раздела. Параметры, задающие размеры отступов, мы оставили такими же, как в стандартной \TeX овской команде: они достаточно разумны, и незачем их трогать, если на то нет особых причин. Наконец, заметим, что это определение можно помещать *только* в стилевой файл, но не в преамбулу документа, поскольку в нем используется команда с символом `@` в имени.

При модификации стиля оформления разделов не забывайте о такой возможности, как изменение вида, в котором представляется на печати номер раздела. Для этого, как говорилось в разделе VII.2.3, надо переопределить соответствующую `the`-команду. Например, если мы хотим, чтобы номера секций печатались римскими цифрами, то надо еще написать


```
\renewcommand{\thesection}{\Roman{section}}
```

Шестой аргумент команды `\@startsection` позволяет автоматически добавлять текст перед номером раздела. К сожалению, не существует простого способа заставить ЛАТЭХ автоматически добавлять текст *после* номера, хотя такая потребность часто возникает (например, хочется, чтобы в заголовках разделов после номеров стояли точки, чего в стандартных ЛАТЭХовских стилях не предусмотрено). Можно, конечно, заставить ЛАТЭХ ставить эти точки после номеров, написав

```
\renewcommand{\thesection}{\arabic{section}.}
```

но после этого автоматические ссылки на секцию, сгенерированные с помощью команды `\ref`, будут иметь нелепый вид:

В разделе 3. говорится о кош-	В разделе <code>\ref{cats}</code>
ках.	говорится о кошках.

В «русифицирующем стиле», использованном при подготовке этой книги, этот недостаток устранен. См. приложение Б.

Заголовки разделов, определяемых вышеописанным способом, печатаются на отдельной строке (или строках, если в одну строку заголовков не умещается). Можно, кроме того, определить раздел, заголовок которого не размещается на отдельной строчке, а просто начинается первый абзац раздела (в нашей книжке так оформлены заголовки самых мелких подразделов, например, раздел 6.5В главы III; в стандартных стилях так оформляются разделы `\paragraph` и `\subparagraph`). Для этого надо пятый аргумент команды `\@startsection` сделать отрицательным. Само значение этого пятого аргумента означает при этом (после отбрасывания знака минус, естественно) величину дополнительного горизонтального отступа между заголовком раздела и продолжающим его текстом из первого абзаца раздела. Например, в стилевом файле, использованном в этой книге, разделы, оформляемые подобно разделу 6.5В главы III, создаются командой `\subsubsection`; соответственно, эта команда переопределена в стилевом файле следующим образом:

```
\renewcommand{\subsubsection}{\@startsection
  {subsubsection}{3}{\parindent}{3.25ex plus 1ex minus
  .2ex}{-.5em}{\normalsize\bf}}
```

Заметьте, что третий аргумент (отступ заголовка от левого поля) на сей раз равен не нулю, но `\parindent`: мы хотим, чтобы первый абзац нашего мелкого раздела, начинающийся с заголовка, имел тот же отступ, что и прочие абзацы. Размер `-.5em` в пятом параметре `\@startsection` взят не с потолка: именно такой отступ заголовка от текста задается в стандартных ЛАТЭХовских командах `\paragraph` и `\subparagraph`, предусматривающих заголовков как составную часть абзаца.

Как видите, для того, чтобы модифицировать оформление разделов, надо знать размеры стандартных ЛАТЭХовских параметров оформления, наподобие отступа перед или после заголовком. Все эти параметры содержатся в стандартных ЛАТЭХовских стилевых файлах, но если Вы не владеете языком ТЭХ, то разобраться в них может быть непросто. На этот случай в приложении Г собраны некоторые важные фрагменты стандартных стилевых файлов «в переводе на ЛАТЭХ»: в виде, который должен быть более понятен читателю этой книги.

Оформление глав отличается от оформления остальных разделов тем, что слово «Глава» и номер главы печатаются на отдельной строчке. С помощью команды `\@startsection` определить такой раздел нельзя, поэтому главы определяются в ЛАТЭХовских стилях 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 эту команду и надо переопределять в своем стилевом файле. Рассмотрим, как `\@makechapterhead` определяется в стандартных стилях. Этой команде передается один аргумент — заголовок главы. В переводе с \TeX на \LaTeX определение выглядит так (не забудьте, что `#1` — это аргумент, то есть текст заголовка).

```
\newcommand{\@makechapterhead}[1]{% Начало макроопределения
  \vspace*{50 pt}% Пустое место вверху страницы.
  {\parindent=0pt
   \raggedright \huge\bf
   \@charapp{} % \@charapp печатает слово "Глава" (см. ниже)
   \thechapter \par % номер главы - в отдельной строке
   \vspace{20pt} % между словом "Глава" и ее заголовком
   \Huge \bf #1\par % заголовок главы
   \pagebreak      % 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 строчку, надо предусмотреть, какие будут при этом параметры верстки абзаца. Как видите, устанавливается нулевое значение абзацного отступа и верстка без 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 `\@charapp` 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. стр. 104). Если в Вашем тексте все главы называются одинаково, то при переопределении `\@makechapterhead` можно не мудрить, а прямо заменить `\@charapp` на слово Глава (или какое-нибудь другое — в зависимости от того, как Вы хотите называть самые крупные разделы текста). Не забудьте только оставить пробел между этим словом и номером главы (в вышеприведенном определении это сделано с помощью обычного трюка с парой фигурных скобок).

Остальное в вышеприведенном определении разъяснений, надо думать, не требует. Скорее всего, Вы захотите в этой команде изменить шрифт, которым печатается заголовок главы, или же интервалы, отделяющие заголовок от остального текста. Чтобы изменить вид, в котором представляется на печати номер главы, надо, как водится, переопределить команду `\thechapter`. Можно также, при желании, изменить вид абзаца, в котором печатается заголовок, установив для него значения параметров `\hangindent`, `\hangafter` или (кто знает?) `\parshape`, можно также задать какое-нибудь более сложное оформление главы с помощью блоков и линеек — все зависит от Вашей фантазии и вкуса!

Надо еще сказать, что мы немного обманули читателя: на самом деле в стандартных стилях команда `\@makechapterhead` определена таким образом, что, если значение счетчика `secnumdepth` отрицательно, то команды, записанные в строчках с пятой по седьмую этого определения, не исполняются (то есть номер главы не печатается). В нашем «переведенном

на Л^AT_EX» определении эти команды будут исполняться всегда, вне зависимости от значения `secnumdepth`; если Вы переопределяете `\@makechapterhead` и не хотите, чтобы главы нумеровались, Вам придется просто удалить соответствующие строки из определения.

За оформление заголовка главы, определенной командой `\chapter` со звездочкой, отвечает команда `\@makeschapterhead`. Ее определение в стандартных стилях аналогично определению `\@makechapterhead`, с тем отличием, что из него удален фрагмент, отвечающий за печать номера:

```
\newcommand{\@makeschapterhead}[1]{% Начало макроопределения
  \vspace*{50 pt}%
  {\parindent=0pt \raggedright
   \Huge \bf #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 стиле `report`), либо с новой страницы, расположенной на развороте справа (как в стиле `book`); у Вас может возникнуть желание повлиять на этот выбор. Во-вторых, по умолчанию абзацный отступ в первом абзаце главы подавляется; Вы можете захотеть сделать так, чтобы он *не* подавлялся. Чтобы решить обе эти проблемы, надо переопределять уже саму команду `\chapter`, а для этого надо знать, как она определяется в стандартных стилях. Вот соответствующее определение, опять в переводе на Л^AT_EX с T_EXa, в стиле `book`:

```
\newcommand{\chapter}{\cleardoublepage
  \thispagestyle{plain}%
  \global\@topnum=0
  \@afterindentfalse
  \secdef\@chapter\@schapter}
```

Разбирать это определение мы не будем, поскольку в нем встречаются некоторые слишком хитрые для читателей этой книги T_EXовские и Л^AT_EXовские конструкции, а просто скажем две вещи:

- Чтобы глава начиналась не обязательно с правой страницы разворота, надо переопределить команду `\chapter`, заменив в этом определении `\cleardoublepage` на `\clearpage`.
- Если Вы не хотите, чтобы подавлялся абзацный отступ в первом абзаце главы, надо переопределить команду `\chapter`, заменив в этом определении `\@afterindentfalse` на `\@afterindenttrue`.

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 не пытаться переопределять `\part` в стилевом файле, а просто оформить соответствующие страницы-заголовки вручную. В конце концов, вряд ли в Вашем тексте будет больше двух-трех таких разделов.

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 \TeX а этот `toc`-файл считывается (с помощью команды `\input`), команды, записанные в него, исполняются, и в результате происходит фактическая верстка оглавления. Аналогичным образом составляется список иллюстраций и список таблиц (при этом информация записывается в файлы с расширениями `.lof` или `.lot` соответственно). Давайте научимся влиять на этот процесс.

Сначала расскажем, как составлять оглавление полностью вручную, игнорируя его автоматическую сборку, обеспечиваемую командами типа `\section`. Итак, предположим, что все команды `\section`, `\chapter` и тому подобные даны в исходном тексте в варианте со звездочкой, и посмотрим, как можно самому создать оглавление.

Для того, чтобы записать в `toc`- (соответственно, `lof`- или `lot`) файл любой текст и любые \TeX овские команды, служит команда `\addtocontents`. У этой команды два обязательных аргумента. Первый из них должен быть `toc`, `lof` или `lot`, в соответствии с тем, в какой из файлов с оглавлениями Вы пишете свой текст. Второй аргумент — текст и команды, которые Вы хотите записать в файл. Если, например, Вам взбрело в голову сделать так, чтобы в автоматически сгенерированное оглавление к Вашей книге попал текст «У попа была собака» (не будем выяснять, зачем), то Вы можете написать:

```
\addtocontents{toc}{У попа была собака.}
```

Если после этого Вы запустите \TeX два раза, то 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 не только записывать в оглавление 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. Подобным образом можно записать в `toc`-файл любую последовательность \TeX овских команд, но при этом необходимо защищать хрупкие команды с помощью команды `\protect` (см. стр. 111 по поводу этой команды и понятия «хрупкая команда»). В случае с `\hrule` мы обошлись без `\protect`, так как эта команда не хрупкая, но вообще, если есть сомнения, то лучше команду защитить. Напомним, что `\protect` действует только на непосредственно следующую команду и что команды смены шрифта в защите с помощью `\protect` не нуждаются. Приведем пример более разумного применения `\addtocontents`, в котором заодно приходится пользоваться и `\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 бы мы, однако, забыли про `\protect`, то получили бы весьма непонятное сообщение об ошибке.

Для того, чтобы составить полноценное оглавление, надо иметь возможность записать в `toc`- (соответственно, `lof`- или `lot`-) файлы 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 первый аргумент в команде `\addcontentsline` был `abcd`, то, когда при следующем запуске ЛАТЭХа будет читаться `toc-` (соответственно, `lof-` или `lot-`) файл, будет исполнена команда `\l@abcd` с двумя аргументами, первый из которых — текст, записанный нами в третьем аргументе команды `\addcontentsline`, а второй — номер страницы, на которую попала Ваша команда `\addcontentsline`. Например, если в Вашем файле было написано

```
\addcontentsline{toc}{abcd}{0 слонах} (*)
```

и если эта команда попала на страницу 95, то при следующем запуске ЛАТЭХа 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 файле должно присутствовать ее определение. Если мы хотим, чтобы запись (*) в исходном файле порождала в оглавлении запись вида

```
О слонах.....95
```

то в своем стилевом файле нам 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` стоят «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`, и так далее. Точнее говоря, если команда для создания разделов документа определена с помощью `\@startsection` (см. раздел 3), то в качестве второго аргумента для `\addcontentsline` выступает «внутреннее имя», под которым ЛАТЭХ знает этот тип разделов документа (то самое, что передавалось 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

так далее. Для модификации строк оглавления, соответствующих `\chapter` или `\part`, надо переопределять команды `\l@chapter` или `\l@part` соответственно. Чтобы было понятно, как переопределять эти команды, рассмотрим, как они определены в стандартных стилях.

В стиле `book` команда `\l@section` определена так:

```
\newcommand{\l@section}%
{\@dottedtocline{1}{1.5em}{2.3em}}
```

Смысл трех аргументов команды `\@dottedtocline` таков. Первый аргумент — «уровень вложенности» элемента оглавления. Если этот уровень превышает значение счетчика `tocdepth`, то команда `\@dottedtocline` ничего в оглавлении не печатает. Второй аргумент — отступ строки оглавления от левого поля. Третий аргумент, также представляющий собой длину, определяет, сколько места в строке оглавления TeX отведет на номер раздела. Результат выводит на печати следующим образом. После отступа, указанного во втором аргументе, печатается номер раздела, затем, отступя от *начала* этого номера столько, сколько сказано во втором аргументе, 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, если раздел имеет слишком длинный номер, то в оглавлении он может наложиться на заглавие. Средство против этого — переопределить в стилевом файле команду `\l@section` (или `\l@subsection...`), увеличив должным образом третий 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 стилевом файле, естественно, так как в имени этой команды присутствует `@`). В стиле `book` эта команда определена как

```
\newcommand{\@pnumwidth}{1.55em}
```

и соответственно на номер страницы отводится 1,55 em места. Если мы хотим, чтобы на номер страницы отводилось 2 em, надо написать в своем стилевом файле

```
\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 3 em, напишите в своем стилевом файле так:

```
\renewcommand{\@tocrmarg}{3em}
```

В этом месте необходимо предупредить читателя об опасности: хотя `\@pnumwidth` и `\@tocrmarg` используются для задания размеров, они *не* являются параметрами со значением длины, и запись наподобие `\@tocrmarg=4em` не даст ничего, кроме сообщения об ошибке!

Наконец, регулировать густоту точек-«лидеров» можно, если переопределить команду `\@dotsep`. В стиле `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):

```
\newcommand{\@dotsep}{3,9}
```

Только не пытайтесь писать вместо этого `\@dotsep=3,9`: получите сообщение об ошибке, и ничего более.

В приложении Г Вы найдете определения команд `\l@section` и им подобных, чтобы Вам было от чего отталкиваться при их модификации.

Теперь рассмотрим, как в стандартных стилях определяются записи в оглавлении, соответствующие самым крупным разделам. Вот (в адаптированном виде, как водится) определение команды `\l@chapter` из стандартного стиля `book`. Чтобы было понятно, о чем идет речь, напомним, что в этом определении на место `#1` подставляется информация о номере (если главы нумеруются) и заглавии главы, а на место `#2` — номер страницы.

```
\newcommand{\l@chapter}[2]%
{\pagebreak[3]\vspace{1em plus 1pt}%
  \@tempdima=1.5em % место для номера главы
  {% Дальнейшее происходит внутри группы...
    \rightskip=\@pnumwidth % см. ниже
    \leftskip=\@tempdima % см. ниже
    \bf % установить жирный шрифт
    \noindent % начать абзац без отступа...
    \hspace{-\leftskip}% и с левого поля.
    #1\nolinebreak\hfil\nolinebreak
    \rlap{\mbox[\@pnumwidth][r]{#2}}\par
    \nopagebreak[3]%
  }% конец этой группы
}% конец определения
```

Определение, как видите, длинное и сложное, к тому же автору не удалось полностью изгнать из него не упоминавшиеся ранее Т_ЕХовские конструкции. Приведено оно здесь не для того, чтобы Вы самостоятельно создавали подобные определения «с нуля» (для этого надо знать про Т_ЕХ больше, чем написано в этой книге), но чтобы Вы при необходимости смогли в нем кое-что осторожно изменить. Чтобы было понятно, что и как менять, разберем это определение по порядку. В начале определения встречаются не рассматривавшиеся нами параметры `\leftskip` и `\rightskip`. 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. При этом `\rightskip` устанавливается равным длине, записанной в определении команды `\@pnumwidth`, определяющей, как мы помним, сколько места будет отведено на номер страницы, а `\leftskip` устанавливается равным значению переменной со значением длины `\@tempdima`, определяющей, сколько места будет отведено на номер главы. Между `\@pnumwidth` и `\@tempdima` есть очень существенная разница. Команда `\@pnumwidth` всегда определяет только место, отводимое на номер страницы, и эту команду можно переопределять в своем стилевом файле. С другой стороны, параметр `\@tempdima` используется Л^AT_ЕXом 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 Л^AT_ЕXовских команд. Поэтому присваивать ему

какое-то значение в стилевом файле совершенно бессмысленно — все равно оно после этого сто раз изменится. Как мог заметить читатель, значение этому параметру присваивается в начале исполнения команды `\l@chapter`, именно это значение принимается в расчет в дальнейшем. Поэтому, если Вы захотите отводить на номер главы, скажем, 2 em вместо 1,5 em, то Вам придется переопределить в своем стилевом файле команду `\l@chapter`, заменив в этом определении третью строчку на

```
\@tempdima=1.5em
```

Такое переопределение `\l@chapter` на практике встречается чаще всего. Например, нужда в нем возникает, если мы переопределяем команду `\thechapter` таким образом, чтоб 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` печатается¹ такой текст (мы предполагаем, что глава называется «Все о слонах» и что ее номер печатается как 5):

```
\makebox[\@tempdima][1]{5}Все о слонах
```

Таким образом, заглавие главы всегда печатается на расстоянии `\@tempdima` от левого поля; если номер главы занимает больше места, чем `\@tempdima`, то он наложится на заглавие.

Команда `\hfil` в десятой строчке обеспечивает пробел между заглавием главы и номером страницы. Если Вы хотите заполнить этот пробел лидерами, можете переопределить команду `\l@chapter`, заменив `\hfil` на, скажем,

```
\leaders\hbox to .5em{\hss.\hss}\hfil
```

(автор не гарантирует, что именно при таком выборе параметров лидеры будут выглядеть красиво).

Надо еще сказать про команды, определяющие вид записей в списке иллюстраций (соответствующих плавающим иллюстрациям) и списке таблиц (соответствующих плавающим таблицам). Они называются `\l@figure` и `\l@table` соответственно, и определяются в стандартных стилях с помощью `\@dottedtocline`. Соответствующие определения приведены в приложении Г.

До сих пор речь шла про сборку 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 знакомая нам команда, которую при работе с русскими текстами приходится переопределять (см. стр. 104). Как видите, заголовок оглавления оформляется просто как заголовок нумерованной секции. Вы можете вместо этого оформить заголовки, скажем, с помощью `\subsection`, или еще каким-либо образом. Новой для Вас будет команда `\@starttoc`. У этой команды предусмотрен один обязательный аргумент. Этим аргументом должно 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. Как определены эти команды в других стилях, написано в приложении Г.

¹Начиная с левого поля, невзирая на установленное значение `\leftskip`; именно для этих целей в определении включена команда `\hspace{-\leftskip}`.

На самом деле в определении `\tableofcontents` присутствует еще команда, позволяющая задать текст для включения в колонтитулы (вспомним, что `\section*` сама по себе никакой информации для колонтитулов не дает). Мы не будем вдаваться в скучные подробности по поводу этой опущенной нами команды. Когда, по прочтении раздела 6, Вы научитесь задавать такие команды, Вы сможете соответствующим образом переопределить и `\tableofcontents`.

5. Перечни общего вида

Теперь мы, наконец, можем завершить наш рассказ о том, как менять стиль оформления перечней (см. раздел VII.2.5).

Начнем с того, что опишем параметры, влияющие на расположение элементов перечня относительно друг друга и остального текста.

5.1. Параметры, влияющие на оформление перечней

Начнем с важного предупреждения. Чтобы отойти от стандартного оформления перечней, необходимо, естественно, изменить значение каких-то из перечисляемых в этом разделе параметров. Однако же, если Вы попытаете попросту присвоить этим параметрам новые значения в своем стилевом файле или в преамбуле, то можете с удивлением обнаружить, что действия это не возымело. Поэтому, если уж Вы начали читать этот раздел, дочитайте его, пожалуйста, до конца: там описаны специальные средства, которые надо применять, чтобы эти изменения возымели действие.

Теперь договоримся о терминологии. Каждый перечень ЛАТЭХ рассматривает как состоящий из *элементов* (каждый элемент вводится, как мы помним, командой `\item`). В свою очередь, каждый элемент перечня может состоять из нескольких абзацев. Наконец, у каждого элемента перечня есть свой *заголовок* — «горошина» на первом уровне окружения `itemize`, заданный Вам заголовок в окружении `description`, и т. п.

Вооружившись этими терминами и имея в виду предупреждение, приступим к утомительному перечислению параметров. Все они — параметры со значением длины. Во-первых, параметры `\leftmargin` и `\rightmargin` задают, с каким отступом от левой (соответственно, правой) границы текста начинается (соответственно, 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 ЛАТЭХ сначала пытается поместить заголовок в блок шириной `\labelwidth`. Если места хватает, то именно в такой блок он и помещается, причем прижатым к правому краю: правый край блока при этом находится на расстоянии `\labelsep` от начала текста, составляющего элемент перечня (так что его левый край будет на расстоянии

$$\leftmargin - \labelwidth - \labelsep \quad (*)$$

от левой границы основного текста или объемлющего перечня). Если же ширина заголовка больше, чем `\labelwidth`, то заголовок печатается как есть (такое, например, регулярно слу-

чается при пользовании окружением `description`); при этом начинаться он будет по-прежнему на расстоянии от левой границы основного текста, определяемом по формуле (*).

Не следует забывать, что при выяснении того, укладывается ли заголовок в размер `\labelwidth`, \TeX смотрит не на то, сколько места он реально займет на печати, а на то, сколько места он занимает с \TeX овской точки зрения. Вот пример:

Науке известны следующие морские животные:	Науке известны следующие морские животные:
Каракатица: Воспета Корнеем Чуковским в сказке «Тараканище».	<code>\begin{description}</code>
Осьминог: ансамблем «Битлз» на диске «Abbey Road».	<code>\item[Каракатица:]</code>
	Воспета Корнеем Чуковским в сказке
	<code>\лк Тараканище\пк.</code>
	<code>\item[\rlap{Осьминог:}]</code>
	Воспет ансамблем <code>\лк Битлз\пк{}</code>
	на диске <code>\лк Abbey Road\пк.</code>
	<code>\end{description}</code>

Мы не сказали еще об одном параметре, влияющем на размещение заголовков. Именно, если параметр `\itemindent` отличен от нуля, то каждый заголовок перечня будет дополнительно сдвинут на это расстояние вправо. Соответственно, при определении, на каком расстоянии начинается заголовок элемента перечня, надо будет прибавить значение `\itemindent` к тому, что получается по формуле (*). По умолчанию значение этого параметра равно нулю.

Если элемент перечня состоит из нескольких абзацев, то по умолчанию во всех этих абзацах абзацный отступ будет отсутствовать. Можно, однако, при желании задать такой режим, что во всех, кроме первого, абзацах каждого элемента перечня будет присутствовать абзацный отступ. Для этого надо задать ненулевую величину этого отступа в параметре `\listparindent`. Кстати, значение этого параметра может быть и отрицательным (в этом случае эффект будет похож на тот, что достигается в обычном тексте установкой параметров `\hangindent` и `\hangafter`).

Параметры, о которых шла речь до сих пор, относились к размещению материала по горизонтали. Теперь займемся «вертикальными» параметрами. Сразу отметим, что все эти параметры являются «растяжимыми» длинами (стр. 80), то есть 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. стр. 81).

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 стилях, то различные элементы перечня будут более отделены друг от друга, чем абзацы внутри одного элемента перечня.

Теперь настало время объяснить, как именно можно менять вышеописанные параметры. При «входе» в перечень \TeX в первую очередь вычисляет уровень вложенности перечней: если перечень не вложен ни в какой другой, то этот уровень равен 1, для перечня, вложенного в перечень, уровень равен 2, и т. д. После этого исполняется команда `\@listI`, если уровень

равен 1, `\@listii`, если уровень равен 2, и т. д.: имя команды — слово `@list`, к которому добавлен уровень вложенности, записанный римскими цифрами (если уровень вложенности равен 1, то римская цифра записывается *прописной* буквой I, в остальных случаях римские цифры записываются *строчными* латинскими буквами). В стандартных стилях все эти команды `\@listI`, `\@listii` и т. п. определены таким образом, что они устанавливают значение параметров оформления перечня на соответствующем уровне. Покажем, как это делается в стандартных стилях при условии, что основным шрифтом имеет кегль 10 (в приложении Г Вы найдете определения этих команд для остальных кеглей).

Во-первых, все стандартные стили определяют параметры со значением длины `\leftmargini`, `\leftmarginii` и т. д. (до `\leftmarginvi` включительно) — в дальнейшем это будут значения `\leftmargin` для перечней на соответствующих уровнях вложенности. Далее, этим параметрам присваиваются значения (например, `\leftmargini` устанавливается равным 25 пунктам, а `\leftmarginii` — 22 пунктам). Затем устанавливаются значения параметров `\labelsep`, `\labelwidth` и `\leftmargin` следующим образом:

```
\leftmargin=\leftmargini
\labelsep=5pt
\labelwidth=\leftmargini
\addtolength{\labelwidth}{-\labelsep}
\setlength{\partopsep}{2pt plus 1pt minus 1pt}
```

После этого, наконец, определяются и команды от `\@listI` до `\@listvi`. Вот, например, как определяется `\@listii`:

```
\newcommand{\@listii}{\leftmargin=\leftmarginii
  \labelwidth=\leftmarginii
  \addtolength{\labelwidth}{-\labelsep}%
  \setlength{\topsep}{4pt plus 2pt minus 1pt}%
  \setlength{\parsep}{2pt plus 1pt minus 1pt}%
  \itemsep=\parsep}
```

Теперь при каждом входе в перечень на уровне вложенности 2 будет исполняться команда `\@listii`, устанавливающая значения параметров `\leftmargin` и прочих (заметим, что `\labelsep` будет равно тем же 5 пунктам: команда `\@listii` нового значения этому параметру не присваивает). Иными словами, некоторые из параметров оформления перечней устанавливаются в стилевом файле обычным порядком, но их большая часть устанавливается заново командами типа `\@listi` при каждом входе в перечень (в следующем разделе мы расскажем, как можно изменить эти автоматически устанавливаемые значения уже в самом перечне).

Итак, чтобы менять оформление перечней, надо переопределять команды `\@listI`, `\@listii`, ..., `\@listvi`. При желании, разумеется, можно сделать и так, что все параметры не будут зависеть от глубины вложенности перечня: для этого надо присвоить им всем какие-то значения в стилевом файле, после чего переопределить команды `\@listI` с `\@listi`, `\@listii` и т. д. на «ничего не делать». Возможен, наконец, и смешанный подход: части параметров присвоить значения сразу, в стилевом файле, поручив заботиться об остальных командам типа `\@listI`. Именно такой подход и принят в стандартных стилях.

5.2. Окружения `list` и `trivlist`

Все \LaTeX овские перечни являются на самом деле частными случаями одной общей конструкции — окружения `list`. Рассмотрим, как это окружение работает.

Первое, что необходимо усвоить: окружение `list` имеет два обязательных аргумента. Поэтому общий вид окружения `list` в исходном тексте будет такой:

```
\begin{list}{заголовок_по_умолчанию}{команды}
.....
\end{list}
```

Параметры окружения `list` означают следующее. *заголовок_по_умолчанию* — это заголовок элемента перечня, печатающийся в том случае, когда этот элемент перечня вводится командой `\item` без аргумента. Например, если бы мы хотели каждый элемент перечня вводить словом «Эх!», то можно было бы написать так:

<p>Эх! Хорошо в стране советской жить.</p> <p>Эх! Хорошо все книжки прочитать, все рекорды мира перегнуть.</p>	<pre>\begin{list}{Эх!}{} \item Хорошо в стране советской жить. \item Хорошо все книжки прочитать, все рекорды мира перегнуть. \end{list}</pre>
--	--

В вышеприведенном примере аргумент *команды* оставлен пустым. Вообще же говоря, это — те команды, которые будут исполнены после входа в перечень и *после* исполнения команды `\@listI` или ей подобной. Поэтому в *командах* можно задать команды, присваивающие новые значения параметрам оформления перечня, отличные от тех, которые задаются командами типа `\@listI`. Кроме этого, в *командах* можно поместить команду `\usecounter`. Эта последняя команда 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 `\newcounter`, 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 использования `\usecounter`:

<p>Вот как выглядят первые буквы латинского алфавита:</p> <p>А: Выглядит так же, как соответствующая русская буква, и читается так же.</p> <p>В: Читается не так, как похожая на нее русская буква.</p> <p>С: И с ней та же история.</p>	<p>Вот как выглядят первые буквы латинского алфавита:</p> <pre>\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.</pre>
--	---

Если Вы хотите, чтобы заголовки элементов перечня выравнивались по левому краю, а не по правому, то можно завершить «заголовок по умолчанию» командой `\hfill`; чтобы по левому краю выравнивались заголовки, заданные в явном виде в необязательном аргументе команд `\item`, надо завершить командой `\hfill` этот необязательный аргумент.

Разумеется, если Вы уж пользуетесь окружением `list`, то разумно это делать не так, как в вышеприведенных примерах, выписывая всякий раз аргументы, а определять через него окружение с помощью `\newenvironment`.

Вот, например, как в стандартных стилях 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. стр. 87).

Наряду с окружением `list`, в ЛАТЭХе 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; `\parsep` равно `\parskip`.
- При входе в это окружение команда `\@listI` или ее аналоги *не* исполняется.
- Команда `\item`, употребленная внутри этого окружения, обязана иметь аргумент (хотя бы пустой).

У Вас может возникнуть вопрос, кому нужны такие «перечни». Ответ: в них сохраняются такие важные черты перечней, 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. Практически поэтому `trivlist` обычно применяют не сам по себе, а для определения новых окружений; при этом в «открывающие команды» `\newenvironment` добавляют команду `\item[]`, а внутри окружения `\item` вообще не используют. Иногда используют и `\item` с аргументом (пример тому Вы увидите ниже, в разделе 7.1).

6. Колонтитулы

Страницу документа, подготовленного с помощью ЛАТЭХа, можно рассматривать как состоящую из трех частей: тела страницы, верхнего колонтитула и нижнего колонтитула. Мы знаем, что с помощью команды `\pagestyle` на оформление колонтитулов можно в какой-то мере влиять. Возможностей, для этого, однако же, не слишком много. Сейчас мы увидим, каким образом радикально менять вид колонтитулов.

Первое, что следует уяснить: если в своем стилевом файле Вы изменили оформление колонтитулов, пользуясь изложенными ниже рецептами, а после этого в тексте использовали команду `\pagestyle`, то она Ваше оформление отменит и установит вместо него стандартное². Предполагая, что Вы имеете в виду это предупреждение, рассмотрим, как оформлять колонтитулы самостоятельно.

²В конце этого раздела рассказано, как можно совместить `\pagestyle` со своим собственным оформлением колонтитулов

Команды, задающие вид верхних колонтитулов, называются `\@oddhead` и `\@evenhead`. Точнее говоря, если стиль оформления документа «двусторонний» (то есть либо задана стилевая опция `twoside`, либо двустороннее оформление подразумевается в основном стиле — см. таблицу на стр. 98), 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 для `\@oddfont` и `\@evenfont`, отвечающих за нижние колонтитулы. Все четыре названные команды получают некоторое определение в стандартных L^AT_EXовских стилях, так что в Вашем стилевом файле переопределять их надо с помощью `\renewcommand`.

Теперь рассмотрим конкретно, как именно надо переопределять вышеназванные команды, чтобы добиться желаемого оформления колонтитулов. Основной принцип таков. При оформлении страницы верхний колонтитул получается в результате исполнения команды

```
\hbox to\textwidth{\@evenhead}
```

(мы предположили, что стиль двусторонний и что страница четная; в остальных случаях — с очевидными изменениями). Можно сказать, что в каждой из этих команд записан текст и T_EXовские команды, которые при верстке страницы будут подставлены в `\hbox to\textwidth`.

Разберем конкретный пример. Предположим, что мы верстаем роман Л. Н. Толстого «Война и мир»; стиль документа мы выберем двусторонний (допустим, основной стиль у нас `book`). Пусть мы хотим, чтобы разворот книги выглядел так:

- На левой странице разворота в верхнем колонтитуле написано имя автора (прижатое вправо).
- На правой странице разворота в верхнем колонтитуле написано название романа (прижатое 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{\@evenfont}{\hfil --- \thepage ---\hfil}
\renewcommand{\@oddfont}{\hfil --- \thepage ---\hfil}
```

Мы здесь предполагали, что страницы с четными номерами будут на левой стороне разворота, а страницы с нечетными номерами — на правой. Так оно и будет, если у документа имеется титульный лист.

Если мы хотим, чтобы каких-то колонтитулов вообще не было, то надо переопределить соответствующую команду на «ничего не делать», например, так:

```
\renewcommand{\@oddfont}{}
```

Прежде, чем перейти к более сложным вещам, связанным с колонтитулами, 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 внести дополнительный элемент в оформление «Войны и мира»: отделять верхние колонтитулы от текста линейками. Как это можно было бы сделать? Разумный выход — 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`, то в голову приходит вот что:

```
\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 могло бы случиться и так, что две соседние линейки на развороте находятся на разной высоте (из-за того, что в колонтитуле на одной странице есть буквы вроде *у*, опускающиеся ниже базисной линии, а на другой — нет). Средство от этого недостатка — добавить `\strut` в наш блок:

```
\renewcommand{\@oddhead}%
{\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'e`. Чтобы уйти от этого, надо еще чуть-чуть схитрить:

```
\renewcommand{\@oddhead}%
{\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 `Overfull'ax` 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'a`, с использованием команды `\underline`. Наш способ, однако, более гибок: например, мы можем регулировать толщину линейки, чего с `\underline` не добьешься.

Итак, мы научились создавать собственные колонтитулы. Однако в стандартных ЛАТЭХовских стилях в колонтитулы обычно помещается не только номер страницы, но и информация о том, в каком месте документа мы находимся (например, номер и заглавие текущего раздела). Давайте научимся делать и это.

Чтобы передать в колонтитул какую-то информацию из текста, в ЛАТЭХе используются команды `\markboth` и `\markright`. Разберем, как они работают. Команда `\markboth`, требующая двух обязательных аргументов, заносит в текст пару «пометок» — два фрагмента текста (возможно, с ТЭХовскими 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 прижатое влево слово «Кот», а в правом — прижатое вправо слово «Пес»³. Кстати, если никаких пометок в тексте нет, то как `\leftmark`, так и `\rightmark` дают «пустой» текст.

До сих пор мы молчаливо предполагали, что на каждой странице присутствует только одна пара пометок. Что будет, если таких пар пометок попадет на страницу несколько? Ответ: команда `\leftmark` в этом случае означает левую пометку из *самой верхней* пары пометок, попавших на страницу, а команда `\rightmark` означает правую пометку из *самой нижней* пары пометок, попавших на данную страницу. С другой стороны, если на страницу вообще ни одна пара пометок не попала, то `\leftmark` и `\rightmark` означают соответственно левую и правую пометки из последней пары пометок, встретившихся до этого.

Поясним сказанное примером. Пусть пометки

```
\markboth{x}{y} и
\markboth{z}{t}
```

попали (в указанном порядке) на страницу 1, на страницу 2 вообще никаких пометок не попало, на страницу 3 попали целые три пары пометок:

```
\markboth{u}{v},
\markboth{a}{b} и
\markboth{m}{n}
```

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>

³Точнее 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.

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 стиле документа: основной стиль — `article`, секции и подсекции нумеруются (иными словами, значение счетчика `secnumdepth` больше единицы), действует стилевая опция `twoside` и в преамбуле или стилевом файле была дана команда `\pagestyle{headings}`.

Во-первых, отметим, что в этом случае пометки в текст вставляют только команды `\section` и `\subsection`. Как же именно они это делают? 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` (помните, что на место `#1` будет подставляться аргумент, т. е. вариант заглавия, идущий в колонтитулы):

```
\newcommand{\sectionmark}[1]{\markboth{%
\uppercase{\thesection\hspace{1em}#1}}% левая пометка
}% правая пометка (она пуста)
}% конец макроопределения
```

Здесь используется TeXовская команда `\uppercase`, которую мы ранее не рассматривали. Не вдаваясь в (довольно хитрые) подробности, скажем, что эта команда переводит все буквы в тексте, попавшем в ее аргумент, из строчных в прописные. Коль о том зашла речь, отметим, что есть еще и команда `\lowercase`, которая наоборот переводит все буквы в тексте, попавшем в ее аргумент, из прописных в строчные. Не пытайтесь, пожалуйста, использовать `\uppercase` и `\lowercase` вне аргументов `\markboth` или `\markright`, иначе Вас подстерегают неприятные сюрпризы.

Команда `\subsectionmark`, определяющая вид пометок, автоматически вставляемых в текст командой `\subsection`, определена следующим образом:

```
\newcommand{\subsectionmark}[1]{\markright
{\thesubsection\hspace{1em}#1}%
}% конец макроопределения
```

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 секции, 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 подсекции (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. Команда `\@evenhead` в этом случае определена так:

```
\newcommand{\@evenhead}{\rm\thepage\hfil\sl\leftmark}
```

Тем самым на страницах с четными номерами (они в данном случае будут левыми на развороте) колонтитул будет выглядеть следующим образом: прижатый влево номер страницы (прямым шрифтом), и заглавие текущей секции (наклонным шрифтом, прописными буквами) — прижатое вправо:⁴ ведь никаких других левых пометок в тексте нет!

Команда `\@oddhead`, отвечающая за верхние колонтитулы на страницах с нечетными номерами, определена в этом стиле так:

```
\newcommand{\@oddhead}{\sl \rightmark\hfil \rm\thepage}
```

Стало быть, верхние колонтитулы к нечетным страницам выглядят так: название текущей подсекции наклонным шрифтом — прижатое влево, номер страницы прямым шрифтом — прижатый вправо. Впрочем, если в текущей секции команд `\subsection` еще не было, то вместо этого 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, как и зачем можно переопределять команды наподобие `\sectionmark`. Предположим, нам не нравится, что в правом колонтитуле иногда бывает пустое место. Чтобы с этим побороться, можно попросить команду `\section` вносить в текст непустую правую пометку — то же заглавие секции. Для этого напишем в своем стилевом файле так:

```
\newcommand{\sectionmark}[1]{\markboth
{\uppercase{\thesection\hspace{1em}#1}}% левая пометка
{\uppercase{\thesection\hspace{1em}#1}}% правая пометка
}% конец макроопределения
```

Теперь, если в разделе нет подразделов, то на страницах с нечетными номерами слева в верхнем колонтитуле будет также печататься заглавие текущей секции. 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 можно распорядиться в этих определениях `\leftmark`'ом и `\rightmark`'ом.

Еще пример. Предположим, Вы вообще не хотите, чтобы при исполнении команды `\subsection` в текст вносились какие-либо пометки. В таком случае можно «отключить» команду `\subsectionmark`, переопределив ее на «ничего не делать»:

```
\renewcommand{\subsectionmark}[1]{}
```

⁴Точнее говоря, это будет заглавие либо текущей секции, либо первой из секций, начинающихся на этой странице — см. выше обсуждение `\leftmark` и `\rightmark`.

Не всегда хочется заниматься столь серьезным делом, как переопределять команды для автоматической расстановки пометок. Бывает, 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 вышеприведенном примере, но \LaTeX предоставляет Вам более простой способ. Если написать в преамбуле

```
\pagestyle{myheadings}
```

то все \LaTeX овские команды для создания разделов не будут вносить пометок в текст, а оформление колонтитулов будет стандартным. Тогда Вы сможете, например, написать

```
\section{0 некоторых специальных свойствах
подмножеств пустых множеств, не рассматривавшихся
в предыдущих разделах статьи}%
\markboth{\thesection\hspace{1em}}%
Подмножества}{}
```

и в оглавлении будет заголовок целиком, а в колонтитуле — всего лишь слово «Подмножества» (мы подразумеваем, что стиль все тот же, только аргументом команды `\pagestyle` был `myheadings` вместо `headings`). Обратите внимание на две тонкости. Во-первых, мы поместили команду, вносящую в текст пометки, *после* команды `\section`, чтобы номер секции, генерируемый командой `\thesection`, был правильным. Во-вторых, мы убрали с помощью знака процента пробел (конец строки) между командами `\section` и `\markboth`, чтобы пометки с гарантией попали на ту же страницу, что и заголовок секции.

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). Мы подробно разобрали, как определены эти команды в одном из вариантов стиля `article`; Вы найдете полный список их определений в приложении Г. 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 тот момент, когда пометки вносятся в текст. Например, если в аргументе `\markboth` присутствует команда `\thesection`, то в пометку реально запишется (и, соответственно, из `\leftmark` или `\rightmark` прочтется) номер секции. Если же необходимо, чтобы какая-то команда записалась в пометку в том же виде, в каком она была задана в аргументе `\markboth` или `\markright`, то надо «защитить» ее, поставив перед ней `\protect`.

Еще один случай, когда пометки в текст вносятся \LaTeX ом автоматически, возникает при оформлении таких фрагментов документа, как оглавление, список иллюстраций или таблиц, список литературы и предметный указатель. Как именно они вносятся, зависит от основного стиля и от того, пользовались ли мы (и как пользовались) командой `\pagestyle`. Именно, если основной стиль — `article`, то никаких пометок при оформлении оглавления и т. п. в текст автоматически вноситься не будет; точно так же не будет этих пометок при любом основном стиле после того, как выполнится команда `\pagestyle` с аргументом `empty`, `plain` или `myheadings`. С другой стороны, если основной стиль — `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 основной стиль — `article`).

Еще одна проблема, которая может возникнуть: как сделать, чтобы в разных частях документа стиль оформления страниц был разным? Типичный случай, когда возникает такая необходимость, таков: хочется, чтобы несколько первых страниц документа (оглавление, предисловие и т. п.) были оформлены стилем `plain` или `empty`. Если написать в начале документа что-нибудь вроде

```
\pagestyle{plain}
```

то, как отмечалось в начале этого раздела, переопределения команд типа `\@oddhead`, сделанные Вами в стилевом файле, пропадут. С другой стороны, в основном тексте документа переопределить эти команды «обратно» будет нельзя, поскольку в их имена входит символ `@`. Выход в том, чтобы дать команду `\pagestyle` *внутри группы!* Когда группа закончится, переопределения, сделанные командой `\pagestyle`, забудутся, и в оставшейся части документа Вы сможете наслаждаться теми колонтитулами, которые сами разработали. Учтите только, что эта группа должна заканчиваться командой вроде `\newpage` или `\clearpage`, иначе может случиться, что стилем, заданным командой `\pagestyle`, оформлено на одну страницу больше, чем нужно.

Итак, Вы теперь имеете широкие возможности для создания собственного оформления колонтитулов. Подведем некоторые итоги. Во-первых, можно написать

```
\pagestyle{myheadings}
```

При этом внешний вид колонтитулов будет стандартный, но материал для колонтитулов Вы будете поставлять вручную, с помощью `\markboth` и/или `\markright`. Во-вторых, можно переопределять команды типа `\sectionmark`: при этом внешний вид колонтитулов будет по-прежнему стандартный, но материал для помещения в колонтитул будет отбираться автоматически по схеме, отличной от стандартной. В-третьих, можно переопределять команды типа `\@oddhead`: при этом Вы меняете как стиль оформления колонтитулов, так и способ обращения с пометками, вносимыми в текст (в частности, если в Ваших определениях команда типа `\@oddhead` не участвуют `\leftmark` и `\rightmark`, то эти пометки будут вообще проигнорированы). Наконец, можно объединить второй и третий подходы. При этом Вы самостоятельно разрабатываете как внешний вид колонтитулов, так и то, какая информация из текста и как будет в них отражена.

7. Разное

7.1. Теоремы, выключные формулы

Чтобы изменить оформление «теорем» (окружений, определяемых с помощью `\newtheorem`), надо переопределить команды `\@begintheorem` и `\@opargbegintheorem` (первая из них отвечает за оформление «теорем» без необязательного аргумента, вторая — за оформление «теорем» с необязательным аргументом. Первая из этих команд определена так:

```
\newcommand{\@begintheorem}[2]{\begin{trivlist}\it
  \item[\hspace{\labelsep}{\bf #1\ #2}]}
```

Здесь аргумент `#1` означает название «теоремы» (например, «Теорема», «Предложение», «Лемма»... — в команде `\newtheorem` это слово являлось вторым обязательным аргументом), а аргумент `#2` означает номер «теоремы». Если мы, например, хотим, чтобы после номера «теорем» стояла точка, нам достаточно переопределить в стилевом файле эту команду, добавив точку после `#1`; что делать для того, чтобы сменить шрифт, которым печатаются номер или текст «теорем», также достаточно ясно.

Команда `\@opargbegintheorem` определяется так:

```
\newcommand{\@opargbegintheorem}[3]{\begin{trivlist}
\it\item[\hspace{\labelsep}{\bf #1\ #2\ (#3)}]}
```

Здесь `#1` и `#2` по-прежнему означают название и номер теоремы, а `#3` — необязательный аргумент «теоремы» (как мы помним, обычно в качестве такового задается имя ученого, которому приписывается данная теорема).

Если Вы не собираетесь делать в оформлении «теорем» более серьезных изменений, чем изменения шрифта и расстановка дополнительных точек, то ничего больше знать про «теоремы» Вам и не надо. Если, однако, Вас не устраивает оформление, задаваемое окружением `trivlist`, то Вы можете переопределить обе эти команды более радикально. Общий принцип тут таков. Перед текстом «теоремы», не имеющей необязательного аргумента, исполняется команда `\@begintheorem`; у этой команды должно быть два аргумента, причем первый из них — название «теоремы», а второй — ее номер. Если «теорема» имеет необязательный аргумент, то перед ее текстом исполняется команда `\@opargbegintheorem`, имеющая три аргумента: первые два — такие же, как у `\@begintheorem`, и третий — необязательный аргумент «теоремы» (имя первооткрывателя). Наконец, *после* текста «теоремы» исполняется команда `\@endtheorem`, которая изначально определена очень просто:

```
\newcommand{\@endtheorem}{\end{trivlist}}
```

В принципе Вы можете переопределить все три эти команды, чтоб получить свое оформление «теорем» (например, в духе макросов для автоматической нумерации задач, о которых шла речь в главе VII). Только следите, чтобы переопределения всех трех команд были согласованы друг с другом: если, например, Вы изгоните `\begin{trivlist}` из определения `\@begintheorem`, но при этом оставите команду `\@endtheorem` в неприкосновенности, то на каждой «теореме» ЛАТЭХ будет сообщать Вам об ошибке (отсутствие баланса `\begin`'ов и `\end`'ов).

Теперь скажем кое-что про стиль оформления номеров выключных формул, заданных в виде окружения `equation`. Помимо таких, уже известных Вам, возможностей, как переопределение команды `\theequation` или изменение подчиненности счетчика `equation` с помощью команды `\@addtoreset`, которые меняют оформление самих номеров формул, можно

также, в принципе, изменить то, что печатается возле этих номеров. Для этого следует переопределить команду `\@eqnnum`. Изначально она определена так:

```
\newcommand{\@eqnnum}{(\theequation)}
```

При желании можно заменить тут круглые скобки на что-нибудь другое. Имейте в виду, что номер выключной формулы обрабатывается Т_ЭХом «в математическом режиме», как формула (латинские буквы по умолчанию набираются «математическим курсивом», и т.п.).

7.2. Сноски

Стиль оформления сносок зависит от многих вещей. Начнем с пробела между страницей и сносками. Чтобы его изменить, надо применить команду `\setlength` с необычным первым аргументом. Вот, например, как выглядит команда, устанавливающая стандартную для Л_АT_ЭXа величину этого пробела:

```
\setlength{\skip\footins}{12pt plus 4pt minus 4pt}
```

То, что пробел сделан растяжимым (см. раздел VIII.3.3), — это понятно, а почему в первом аргументе целых две команды и что они значат, объяснить в рамках этой книги невозможно. Так что воспринимайте этот рецепт для установки пробела между текстом и сносками чисто догматически (кому не лень, может попытаться узнать всю правду из пятнадцатой главы 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 Т_ЭХа, 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 ширину 0,4 пункта), и затем спускаемся на 2,6 пункта вниз. В итоге получается, что и линейка напечаталась, и места по вертикали мы не занимаем, поскольку $-3 + 0,4 + 2,6 = 0$. Если Вы хотите изменить ширину или толщину линейки, команду `\footnoterule` можно переопределить; только не забудьте проследить, чтобы толщина линейки была скомпенсирована отрицательным `\vspace`'ом! Можно, собственно говоря, сделать так, чтобы этой линейки вообще не было, сказав

```
\renewcommand{\footnoterule}{}
```

(уж в этом-то случае можно не сомневаться, что места по вертикали мы не займем!). Если Вам вдруг понадобится задать совсем иной разделитель между сносками и текстом, можете переопределить команду `\footnoterule` и принципиально по-иному. В этом случае необходимо знать следующее:

- Команда `\footnoterule` будет вызываться в те моменты, когда Т_ЭХ находится в вертикальном режиме.

- По окончании работы команды `\footnoterule` Т_ЕX должен снова оказаться в вертикальном режиме.
- Текст, генерируемый командой `\footnoterule`, не должен, с точки зрения Т_ЕXа, 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`'а (см. раздел 9.3 главы III). Так вот, `\footnotesep` задает высоту этой линейки.

За вид номеров сносок в тексте отвечает команда `\@makefnmark`. По умолчанию она определена следующим образом:

```
\newcommand{\@makefnmark}{\hbox{\mathsurround=0pt
\@thefnmark}}}
```

Здесь на место команды `\@thefnmark` при выполнении будет подставлен номер сноски (или то, что его заменяет, если мы пользовались командой `\footnotemark`). Обратите внимание, что номер сноски оформлен как верхний индекс в математической формуле — именно за счет этого 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 (если Вы пропустили раздел про `\mathsurround`, можете проигнорировать это место).

И, наконец, самое главное — команда, генерирующая собственно текст сноски. Она называется `\@makefnmark`. Вот ее стандартное определение, в котором аргумент #1 обозначает текст сноски, а на команда `\@thefnmark` означает то же, что и выше:

```
\newcommand{\@makefnmark}[1]{\parindent=1em\noindent
\hbox to 1.8em{\hss\@thefnmark}\#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.

Если Вы захотите переопределить `\@makefnmark` таким образом, чтобы изменилось оформление номера сноски, например, чтоб он печатался так,¹⁰⁾ то не забудьте изменить оформление номера сносок и в самом тексте, переопределив согласованным образом команду `\@makefnmark`.

7.3. Библиография

Если Вам не нравится, что библиографические ссылки печатаются в квадратных скобках, то Вы можете переопределить в своем стилевом файле команды `\@cite` и `\@biblabel`. По умолчанию они определены так:

```
\newcommand{\@cite}[2]{[#1\if@tempswa , #2\fi]}
\newcommand{\@biblabel}[1]{[#1]\hfill}
```

¹⁰⁾Нет, нет, это только для примера.

Мы не будем даже пытаться объяснить, что означают заковыристые команды в первом из этих макроопределений и как второе из них согласуется со всем остальным, а отметим только, что в первом из этих определений можно заменить квадратные скобки на какие-нибудь другие (скажем, круглые или косые), — и тогда в соответствующих скобках будет печататься ссылка, генерируемая командой `\cite`; аналогично, если заменить скобки во втором из этих определений, то в соответствующих скобках будет печататься номер источника в списке литературы. Будем надеяться, что для Вас уже очевидно, что в *Вашем* стилевом файле надо писать `\renewcommand` вместо `\newcommand`.

7.4. Предметный указатель

Первое, что Вы можете изменить в оформлении предметного указателя (окружение `theindex`), — это отступы, создаваемые командами `\item`, `\subitem` и т. д. Для изменения отступов «на первом уровне» (создаваемых командой `\item`) надо в стилевом файле 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 стандартных ЛАТЭХовских стилях это не предусмотрено); возможно также, что Вы захотите предпослать предметному указателю небольшое введение, набранное во всю ширину страницы. Чтобы добиться таких вещей, надо переопределить команду `\theindex`. Ее стандартное определение в стилях `book` и `report` выглядит так:

```
\newcommand{\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}
```

Первая, вторая и последняя строчки этого определения содержат незнакомые Вам Т_ЕХовские команды; мы не будем пытаться объяснить, что они значат, а только скажем, что менять эти места в определении команды `\theindex` нельзя. Зато все остальное в этом определении

должно быть понятно читателю, усвоившему основную часть нашей книги. Именно, в третьей строчке 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. раздел 2.1 главы IV). В пятой строке дана команда, задающая (неким экзотическим, не рассматривавшимся нами способом) материал для колонтитулов. Она либо вносит левую и правую пометки, совпадающие со стандартным заглавием предметного указателя, либо ничего не делает (в разделе 6 объяснялось, в каких случаях окружение `theindex` вносит автоматически пометки, а в каких — нет). Если уж Вы переопределяете `\theindex`, то можете в этой строчке написать `\markboth` вместо `\@mkboth` (если хотите, чтоб эти пометки были), или вообще убрать эту строчку, 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. (если Вы поняли, что такое `\hangindent`, то поймете, каким именно образом согласовано).

Наконец, в четвертой строчке стоит команда `\twocolumn` с необязательным аргументом. Как объяснялось в разделе 7.6 главы III, то, что стоит в необязательном аргументе, будет напечатано во всю ширину страницы. В стандартном определении тут стоит всего лишь команда `\@makeschapterhead`, создающая заголовок нумерованной главы (см. раздел 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 лазить в стилевой файл. Один из возможных выходов таков. Запишите в Вашем определении команды `\theindex` четвертую строчку так:

```
\twocolumn[\@makeschapterhead{\indexname}\указ]%
```

Здесь `\указ` — команда, которую Вы сможете определить в преамбуле документа, например, так:

```
\newcommand{\указ}{%
```

```
В предметном указателе я собрал  
все непонятные слова. Если Вы  
чего-то в нем не найдете,  
обратитесь к словарю  
иностраннных слов.%  
}
```

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}
```

Последнее, что нужно сказать про команду `\theindex`, — это что в стиле `article` четвертая строчка ее стандартного определения выглядит так:

```
\twocolumn[\section*{\indexname}]%
```

(поскольку в стиле `article` главы не определены).

Приложение А.

О Т_EXовских шрифтах

Большая часть шрифтов, используемых Л^AT_EXом, была создана самим Дональдом Кнудом с помощью им же написанной программы METAFONT. Эти шрифты называются Computer modern fonts. Скажем несколько необходимых для дальнейшего слов о том, в каком виде хранятся Т_EXовские шрифты. Как мы неоднократно отмечали выше, в процессе верстки текста Т_EXу неважно, как реально выглядят символы. Все, что в этот момент используется — информация о том, сколько места надо оставить на каждый символ (плюс еще некоторые тонкости: например, какие символы составляют лигатуру, подобно тому как сочетание ‘*д*’ дает на печати *д*, между какими символами предусмотрен кернинг, и т. п.). Все эти данные содержатся в файлах с расширением `.tfm`. Формат этих файлов входит в Т_EXовский стандарт. Реальная форма символов становится важна при просмотре, печати, и вообще в тех случаях, когда в игру вступают `dvi`-драйверы. Поэтому информация о форме символов хранится в отдельных файлах, формат которых не стандартизирован, а зависит от используемых `dvi`-драйверов. Мы будем для краткости называть файлы, в которых записана форма символов, `pk`-файлами, так как они обычно (хотя и не всегда) имеют расширение `.pk`.

Следующее, что необходимо знать про Т_EXовские шрифты, — это то, что одному `tfm`-файлу может соответствовать много разных шрифтов (и, тем самым, много разных `pk`-файлов). Дело в том, что в Т_EXовских шрифтах, созданных с помощью METAFONTа, предусмотрена возможность их масштабирования. В процессе трансляции исходного текста Т_EX при работе с масштабированным (скажем, увеличенным в два раза) шрифтом просто умножает на заданный коэффициент размеры символов, взятые из уже существующего `tfm`-файла. А вот при печати или просмотре понадобится уже и новый `pk`-файл, соответствующий масштабированному шрифту.

Л^AT_EX предусматривает возможность включения в Ваш документ шрифтов, не входящих в стандартный Л^AT_EXовский комплект. Для этого используется команда `\newfont`. Ее формат таков:

```
\newfont{команда}{описание_шрифта}
```

Здесь *команда* — это команда для переключения на добавляемый Вами шрифт. Имя этой команды, которое Вы должны придумать, подчиняется обычным Т_EXовским правилам и не должно совпадать с именами уже существующих команд. Что же до *описания_шрифта*, то в простейшем виде это — просто имя `tfm`-файла, соответствующего данному шрифту. Вот пример. В свое время Дональд Кнут шутки ради разработал такой причудливый шрифт, называемый `cmff10`. Если Вы хотите пользоваться этим шрифтом в своем тексте, включите в преамбулу строчку

```
\newfont{\косой}{cmff10}
```

и вы сможете писать тексты вроде

Буквы выглядят немного кособо-
кими.

Буквы выглядят немного
{\косой кособокими.}

По сути дела, `\newfont` определяет новую команды для переключения шрифтов, похожую на привычные Вам команды вроде `\bf` или `\Large` (в частности, действие этой новой команды заканчивается по выходе из той группы, внутри которой она была дана). Надо, однако, иметь в виду два существенных различия. Во-первых, в отличие от команд наподобие `\small`, такие команды *не* меняют ни интервалов между строками (параметр `\baselineskip`), ни размеров невидимой линейки, создаваемой командой `\strut`. Во-вторых, внутри математической формулы такая команда вообще никакого действия не оказывает и никак на шрифт, которым печатаются математические символы, не влияет.

Выше говорилось, что в Т_ЭХе можно использовать масштабированные шрифты. Чтобы подключить такой шрифт с помощью команды `\newfont`, надо задать требуемое увеличение или уменьшение во втором аргументе команды `\newfont`. Оно задается с помощью Т_ЭХовского «ключевого слова» `scaled` (без backslash'a!), за которым следует коэффициент масштабирования, умноженный на 1000 (после умножения коэффициента на 1000 должно получиться целое число). Например, для подключения шрифта, увеличенного в два с половиной раза, надо после имени `tfm`-файла написать `scaled 2500`, а для шрифта, размеры которого уменьшены на 20%, надо написать `scaled 800`. Будем надеяться, что у Вас есть либо `pk`-файлы, соответствующие масштабированным таким образом шрифтам, либо программа METAFONT, с помощью которой эти файлы можно сгенерировать (при условии, что Вы располагаете еще и «исходными текстами» шрифтов).

В реально используемых в Т_ЭХе шрифтах коэффициенты 2,5 или 0,8 на самом деле не употребляются. Применяемые на практике коэффициенты увеличения образуют геометрическую прогрессию со знаменателем 1,2. Для таких увеличений можно использовать более удобные обозначения, при помощи команды `\magstep`: вместо `scaled 1200` можно написать `scaled \magstep 1`, вместо `scaled 1440` — `scaled \magstep 2` (поскольку $1,2^2 = 1,44$), и т. д. (максимально возможное значение — `\magstep 5`). Можно также сказать `\magstephalf`, что задает увеличение в $\sqrt{1,2}$ раза.

Можно также задавать увеличение не в явном виде, а сообщить Т_ЭХу требуемый «характерный размер» шрифта. Для этого надо во втором аргументе команды `\newfont` написать, после имени `tfm`-файла масштабируемого шрифта

at размер

где *размер* — требуемый «характерный размер», заданный обычным образом в Т_ЭХовских единицах длины или через Т_ЭХовские параметры длины. Если основной шрифт Вашего документа имеет кегль 10, то характерный размер разумно выбирать равным 10pt, если 11 или 12, то 11pt или 12pt соответственно. Например, существует (не входящий в Л^AТ_ЭХовский стандартный комплект) шрифт `wasy10`, содержащий различные причудливые символы, в том числе астрологические знаки. Если Вы пишете текст по астрологии шрифтом кегль 12, то имеет смысл написать в преамбуле

```
\newfont{\астролог}{wasy10 at 12pt}
```

В заключение приведем таблицы кодов символов в основных текстовых шрифтах, используемых Л^AТ_ЭХом. Необходимость знать эти коды возникает сравнительно редко, например,

когда Вы пользуетесь ЛАТЭХовской командой `\symbol` (см. стр. 60). В таблицах не представлены коды русских букв, поскольку они зависят от используемой русификации. Коды символов даны в шестнадцатиричной записи: первая цифра записи определяет строку, в которой стоит символ, вторая — столбец. Например, в шрифте `gotan` буква \ae стоит на пересечении строки "10 и столбца "0A ; значит, ее код равен 1A в шестнадцатиричной записи и $1 \cdot 16 + 10 = 26$ в десятичной записи, и эту букву можно напечатать командами `\symbol{"1A}` или `\symbol{26}`. Наша первая таблица — шрифт `gotan`, который является текущим в начале работы ЛАТЭХа и вызывается, если надо, командой `\rm`. Соответствующие `tfm`-файлы называются `cmr10`, `cmr11` и т. п.: последние две цифры совпадают с «характерным размером» шрифта, выраженным в пунктах.

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	˘	˙	˚	˛	˜	ˆ	˜	˘	I	■	■
"10	“	”	■	■	˘	—	—		■	ı	ı	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	-

Как видите, наряду с буквами и цифрами в этой таблице присутствуют и диакритические знаки, и лигатуры, и даже некоторые греческие буквы.

Таблиц для наклонного и жирного шрифта мы не приводим, поскольку они не имеют принципиальных отличий от таблицы для прямого шрифта: на одинаковых местах стоят символы с аналогичными очертаниями.

С другой стороны, в курсивном шрифте (напомним, что он вызывается командой `\it` и что на него же в большинстве случаев переключается текущий шрифт под действием команды `\em`; названия `tfm`-файлов `cmi10` и др.) некоторые символы уже выглядят принципиально по-иному:

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	˘	˙	˚	˛	˜	ˆ	˜	˘	I	■	■
"10	“	”	■	■	˘	—	—		■	ı	ı	ff	fi	fl	ffi	ffl
"20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
"30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
"40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
"60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	-

Обратите, в частности, внимание, что на месте знака доллара в курсивном шрифте стоит знак для фунта стерлингов, так что, если Вы скажете `\it\$`, то на печати получится вовсе не доллар. Не забудьте об этом, если Вы будете писать бюллетень курсов иностранных валют.

Наконец, последняя из таблиц шрифтов, которые стоит привести — это таблица шрифта `typewriter` (вызываемого командой `\tt`; `tfm`-файлы называются `cmtt10` и др.). В ней также есть принципиальные отличия от двух предыдущих.

	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F
"00	`	´	^	~	¨	■	°	˘	˙	-	■	د	ç	I	■	■
"10	■	■	■	■	˘	-	--		■	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	{		}	~	-

Приложение Б.

Опция `ruscorr.sty`

В этом приложении мы описываем разработанный автором «русифицирующий стиль» (точнее, стилевую опцию) `ruscorr.sty`, использованный при подготовке настоящей книги. Для пользования этим стилем необходимо иметь файл `ruscorr.sty`. Коль скоро этот файл у Вас есть, для подключения русифицирующего стиля надо указать `ruscorr` в необязательном аргументе команды `\documentstyle`, *после* всех стандартных ЛАТЭХовских стилевых опций, но *до* стилевой опции, предназначенной специально для данного текста, как это объяснялось в главе IX.

Перечислим теперь дополнительные возможности, предоставляемые стилевой опцией `ruscorr`. Во-первых, в этой стилевой опции определены команды для тригонометрических, обратных тригонометрических и гиперболических функций, дающие их обозначения в таком виде, как это принято в России. Эти команды таковы:

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

На стр. 32 объяснялось, как определять аналогичные команды самостоятельно; при пользовании `ruscorr` для девяти вышеперечисленных функций Вам это не понадобится.

Кроме того, определены команды `\eps` и `\vphi`, равносильные ТЭХовским и ЛАТЭХовским `\varepsilon` и `\varphi` соответственно: в русских текстах более принято начертание букв «эпсилон» и «фи», задаваемое именно этими командами, так что есть смысл создать для них и более короткие обозначения.

Далее, при пользовании стилем `ruscorr` по-русски будут печататься различные «стандартные заголовки» наподобие «Глава», «Оглавление», и т. п. Все эти стандартные заголовки перечислены в разделе 3.3 главы IV (как там объясняется, для их русификации достаточно переопределить команды типа `\chaptername`).

При пользовании стилем `ruscorr` большая часть сообщений об ошибках, выдаваемых ЛАТЭХом, будет выдаваться по-русски. Подчеркнем, что это относится только к ошибкам, сообщения о которых начинаются со слов `LaTeX error`, а также к ЛАТЭХовским предупреждениям (например, о том, что метки могли измениться и поэтому надо запустить ЛАТЭХ еще раз); сообщения об ошибках, встроенные в ТЭХ, будут по-прежнему выдаваться по-английски, и тут уж ничего не поделаешь.

Первый абзац главы, секции и т. п. будет делаться, при пользовании стилем `ruscorr`, с абзацным отступом (в стандартных ЛАТЭХовских стилях абзацный отступ в этой ситуации подавляется). В главе IX объяснялось, как добиться этого же эффекта с помощью переопределения некоторых команд в своем стилевом файле. Еще одно изменение в оформлении разде-

лов документа, которое делает стилевая опция `russcorr.sty`, таково: в заголовках разделов документа после номера раздела ставится точка, чего в стандартных ЛАТЭХовских стилях не предусмотрено. На взгляд автора, такое оформление более соответствует нашим традициям. Если Вы хотите, чтобы после номеров стояли не точки, а что-то другое (например, вообще ничего, как в ЛАТЭХовском стандарте), то `russcorr.sty` дает Вам возможность этого добиться, переопределив в преамбуле команды `\postchapter`, `\postsection` и т. п. Например, если Вы начнете свой файл так:

```
\documentstyle[11pt,russcorr]{article}
\renewcommand{\postsubsection}{}
```

то после номеров подсеций точки ставиться *не* будут.

Вы можете также определить команду, которая будет ставить какой-то значок *перед*, а не после номера секции, подсеции и т. п.¹ Для этого надо переопределить команды `\presection` для секции, `\presubsection` для подсеции, и т. п. (по умолчанию стиль `russcorr` определяет их как «ничего не делать», так что никакого дополнительного текста перед номерами не ставится). Например, если Вы хотите, чтобы перед номерами секций печатался знак параграфа, то Вы можете начать файл так:

```
\documentstyle[11pt,russcorr]{article}
\renewcommand{\presection}{\S}
```

Если переопределить `\presection` таким образом, чтобы перед номером раздела шел длинный текст, то результаты будут неудовлетворительны. Если возникнет такая необходимость, лучше переопределить команду `\section`, пользуясь рецептами, изложенными в разделе 3 главы IX.

Проблема с точками возникает, при пользовании стандартными стилями ЛАТЭХа, и в нескольких других местах; во всех этих случаях стиль `russcorr` вносит соответствующие изменения в оформление. Именно, при пользовании этим стилем подписи к «плавающим» рисункам и таблицам будут иметь вид наподобие

Рис. 4.2. Тигр в разрезе.

(в стандартных стилях после номера ставится двоеточие, что в русском тексте выглядит очень неудачно). Далее, точки стоят и после номеров «теорем» (напомним, что так называются окружения, определенные с помощью `\newtheorem`); если Вам больше нравится, чтобы точки в этих местах не ставились, как это и предусмотрено стандартом, Вы можете в преамбуле переопределить команду `\afterthmseparator` на «ничего не делать»:

```
\renewcommand{\afterthmseparator}{}
```

Подчеркнем, что `\afterthmseparator`, так же как и описанные выше команды, указывающие, что ставится после номера или перед номером раздела (`\postchapter` и ей подобные) *не* определены в стандартных ЛАТЭХовских стилях, и переопределять их можно только при пользовании стилевой опцией `russcorr`.

Следующие изменения относятся к счетчикам. При пользовании стилевой опцией `russcorr` становятся определенными команды `\ralph` и `\Ralph`, представляющие собой «русифицированные» варианты стандартных ЛАТЭХовских команд `\alph` и `\Alph`: единственным (обязательным) аргументом этих команд является имя счетчика, и на печать эти команды выдают

¹Но не главы; чтобы задавать текст, идущий перед номером главы, можно и нужно переопределять команду `\chaptername`.

русскую букву номер которой в алфавите равен значению счетчика (буква будет строчной в случае `\ralph` и прописной в случае `\Ralph`; буквы ё, й, ь, ы, ь в счет не включаются).

При использовании `russcorr` высота страницы, принятая по умолчанию, слегка увеличена, в соответствии с рекомендациями А. Шеня, содержащимися в [3]; разумеется, Вы по-прежнему можете (в преамбуле или в стилевом файле, подключаемом после `russcorr.sty`) изменить размеры страницы по своему усмотрению, как это объяснено в разделе 2.2 главы IV.

При использовании стилевой опцией `russcorr` Вы получаете возможность автоматически подсчитывать общее количество страниц в документе (чтобы, например, указать это количество в выходных данных). Чтобы это сделать, надо, написать в преамбуле команду

```
\countpages
```

и после этого счетчик по имени `totalpages` будет содержать общее количество страниц в книге:²

Число страниц в этой книге равно 0.

Число страниц в этой книге равно $\sim\arabic{totalpages}$.

При первом запуске ЛАТЭХа для обработки документа значение этого счетчика равно -1 ; в дальнейшем это значение определяется (подобно ссылкам, генерируемым с помощью `\ref`) по результатам предыдущего запуска. Если в промежутке между двумя запусками количество страниц изменилось, ЛАТЭХ попросит Вас запустить его еще раз.

Титульный лист, имеющий номер 0, в общий счет страниц не идет. Команда `\dontcountpages` отменяет подсчет числа страниц, действуя противоположно команде `\countpages`; в этом случае значение счетчика `totalpages` будет всегда равно -1 . Именно такой режим принят по умолчанию, в отсутствие команды `\countpages`.

Стиль `russcorr.sty` несколько меняет оформление нумерованных перечней. Во-первых, элементы нумерованного перечня (задаваемого, как мы помним, окружением `enumerate`), обозначаются цифрой со скобкой, а не цифрой с точкой, как в стандарте. Во-вторых, если одно окружение `enumerate` вложено в другое, то элементы «внутреннего» перечня нумеруются по алфавиту русскими буквами (а не латинскими, как в стандартных стилях). Если во второе из этих окружений вложено еще одно `enumerate`, то элементы перечня «уровня вложенности три» вообще не нумеруются, а отмечаются тире. Наконец, стиль `russcorr` запрещает вкладывать четыре `enumerate` друг в друга: автор не смог придумать никаких разумных пометок для этого случая, а то, что предусмотрено ЛАТЭХовским стандартом, в русском тексте смотрится неудачно.

Наряду с изменением оформления нумерованных перечней, можно также создавать перечни, элементы которых будут сразу нумероваться русскими буквами по алфавиту. Для этого в стиле `russcorr.sty` предусмотрено окружение `rlist`:

²Точнее говоря, номер последней страницы.

TeX удобен по следующим причинам:

- а) Он не слишком требователен к используемой аппаратуре.
- б) Математические формулы получаются очень хорошо.
- в) TeX — фактический стандарт для международного обмена статьями по математике.

TeX{} удобен по следующим причинам:

```
\begin{rlist}
\item Он не слишком требователен к используемой аппаратуре.
\item Математические формулы получаются очень хорошо.
\item TeX{} --- фактический стандарт для международного обмена статьями по математике.
\end{rlist}
```

Можно также использовать окружение `rlist` с необязательным аргументом: если написать

```
\begin{rlist}[u]
```

вместо

```
\begin{rlist}
```

то элементы перечня будут нумероваться прописными буквами вместо строчных.

При желании можно изменить оформление перечней, задаваемых окружением `rlist`. Для этого надо переопределить команду `\labelrlist`. В стилевой опции `russcorr` она определена так:

```
\newcommand{\labelrlist}{\therlistctr)}
```

Здесь `therlistctr` — `the`-команда, соответствующая счетчику `rlistctr`, который отвечает за оформление перечней `rlist`. Подчеркнем еще раз, что команда `\labelrlist` и счетчик `rlistctr` не определены в стандартном L^AT_EX_E.

При использовании стилевой опции `russcorr` команда `\appendix` (см. стр. 104) устанавливает нумерацию самых крупных разделов документа прописными русскими буквами (вместо прописных латинских в стандартных стилях).

Команда `\сегодня` выдает дату трансляции текста по-русски (в отличие от стандартной команды `\today`).

В стиль `russcorr` добавлена команда для постановки диакритического знака такого вида: \mathring{o} (буквы с такими знаками встречаются в чешском языке). Обозначается эта команда `\oo` (две строчные латинские буквы `o`). Таким образом, чтобы на печати получить \mathring{o} , в исходном файле надо написать `\oo{u}` или `\oo u`.

Наконец, стиль `russcorr` определяет команды для кавычек формы, принятой в России, и знака номера: левая и правая кавычки-„лапки“ обозначаются `\glqq` и `\grqq` соответственно, левая и правая кавычки-«елочки» обозначаются `\лж` и `\пж` соответственно (имена двух последних команд состоят из русских букв!), а знак номера печатается командой `\номер` (имя также состоит из русских букв). Пользоваться последними тремя командами можно только в том случае, если в Вашем распоряжении есть русские шрифты для TeXa, разработанные А. Самариним и Н. Глonti в модификации А. Шеня.

Приложение В.

Новая схема выбора шрифтов (NFSS)

В анонсированной третьей версии \LaTeX а, а также в уже используемом \mathcal{AMS} - \LaTeX е, работа со шрифтами организована не так, как в стандартном \LaTeX е. В этом приложении мы скажем несколько слов об этой организации шрифтов, представляющей собой ближайшее будущее \LaTeX а.

Новая схема выбора шрифтов (по-английски New Font Selection Scheme, сокращенно NFSS) характерна прежде всего тем, что в ней используется больше шрифтов, чем в стандартном \LaTeX е. Именно, каждый шрифт характеризуется четырьмя параметрами, и в принципе возможно переключиться на шрифт, отвечающий любой комбинации этих параметров. Параметры, характеризующие шрифты, таковы: *семейство* (family) — например, roman, сансериф или «имитация пишущей машинки», *серия* (series) — она характеризует как «жирность» шрифта, так и его плотность, то есть размеры интервалов между буквами и словами, *форма* (shape) — например, прямой, курсивный, наклонный... и *размер* (при выборе размера можно также задать и значение `\baselineskip` для данного шрифта). Таким образом, при использовании NFSS можно переключаться и на такие шрифты, как жирный курсив или наклонный сансериф (разумеется, необходимо, чтобы эти шрифты у Вас имелись).

При использовании NFSS не требуется всякий раз при переключении шрифта включать в текст громоздкие команды, задающие все четыре параметра требуемого шрифта: по-прежнему поддерживаются привычные нам команды наподобие `\rm`, `\it`, `\sf` и т. п. К сожалению, смысл этих команд не совсем такой, как в стандартном \LaTeX е. Именно, команды `\rm`, `\sf` и `\tt` изменяют только семейство, команда `\bf` изменяет только серию, а команды `\sl`, `\it` и `\sc` изменяют только форму. В связи с этим один и тот же исходный текст при обработке с помощью стандартного \LaTeX а и NFSS может дать разные результаты. Например, если в файле написано

```
\it собака \rm кошка
```

то в стандартном \LaTeX е собака получится курсивной, а кошка будет напечатана прямым шрифтом. При использовании же NFSS команда `\rm` всего лишь меняет текущее семейство на «стандартное» (противопоставляемое сансерифу и имитации пишущей машинки), которое, скорее всего, к этому моменту и так было текущим, так что переключения на новый шрифт не произойдет и оба домашних животных будут напечатаны одним и тем же курсивом. Чтобы перейти на прямой шрифт после курсива, следовало бы вместо `\rm` дать предусмотренную в NFSS команду `\normalshape`, меняющую текущую форму шрифта на «нормальную». Впрочем, есть и более простой способ: если дать команду `\it` внутри группы, то по выходе из группы сделанные этой командой изменения забудутся и восстановится ста-

тус-кво. При этом Вам не нужно и помнить о том, какими параметрами характеризуется шрифт в NFSS и как эти параметры менять. Иными словами, наш совет таков:

При пользовании NFSS делайте все изменения текстовых шрифтов только внутри групп!

Впрочем, не худо следовать этому совету и при написании текстов на стандартном ЛАТЭХе: при этом Вы облегчите себе жизнь, если придется обрабатывать текст с помощью будущих версий ЛАТЭХа, использующих NFSS.

Еще одно место, в котором проявляется несовместимость между стандартным ЛАТЭХом и NFSS — это использование шрифтов в математических формулах. Тут различия достаточно серьезны. Например, при использовании NFSS нельзя переключаться на прямой шрифт в формуле с помощью команды `\rm` (вместо этого надо писать `\mathrm`). Не надо, конечно, думать, что новшества, привносимые NFSS в набор формул, сводятся только к неудобствам. На самом деле новая схема выбора шрифтов позволяет радикально изменить шрифтовое оформление формул (например, сделать так, что основным шрифтом в формулах будет не математический курсив, а сансериф).

Если, наконец, Вы по тем или иным причинам (например, из-за отсутствия требуемого набора шрифтов) не желаете наслаждаться теми возможностями, которые предоставляет Вам NFSS, то есть возможность набирать тексты «по старинке». Для этого необходимо указать в команде `\documentstyle` стилевую опцию `oldfont`, и тогда все команды смены шрифта будут иметь в точности тот же смысл, что и в стандартном ЛАТЭХе. По крайней мере, в АМ \mathcal{S} -ЛАТЭХе такая возможность предусмотрена; надо думать, нечто аналогичное будет и в версии 3 ЛАТЭХа.

Приложение Г.

Избранные места из стандартных стилей

Это приложение фактически является продолжением главы IX. Как мы уже видели в этой главе, при модификации стандартных стилей полезно знать, как именно в них определены различные параметры оформления (чтоб было от чего отталкиваться). В настоящем приложении собраны различные важные фрагменты стандартных Л^AT_EXовских стилевых файлов, в которых эти определения содержатся. Для удобства читателей определения, как и в главе IX, кое-где переписаны и упрощены: вместо T_EXовских команд используются менее эффективные, но зато знакомые читателю Л^AT_EXовские; там, где в оригинальных текстах использованы T_EXовские «условные макросы», приводятся результаты развертывания этих макросов в зависимости от условий; обозначения для чисел и длин переведены в явную форму, и т. п. По существу приводимые нами макроопределения эквивалентны приведенным в стандартных стилевых файлах.

1. Основные размеры

Начнем с размеров полей. Перечислим, какие значения присваиваются параметрам, отвечающим за поля текста, в зависимости от кегля. В «двусторонних» вариантах стилей `article` и `report` эти параметры таковы:

	Кегль		
	10pt	11pt	12pt
<code>\oddsidemargin</code>	44pt	36pt	21pt
<code>\evensidemargin</code>	82pt	74pt	59pt
<code>\marginparwidth</code>	107pt	100pt	85pt
<code>\marginparsep</code>	11pt	10pt	10pt

Значение перечисленных параметров объясняется в главе IV.

В «односторонних» вариантах стилей `article` и `report` эти параметры устанавливаются так:

	Кегль		
	10pt	11pt	12pt
<code>\oddsidemargin</code>	63pt	54pt	39.5pt
<code>\marginparwidth</code>	90pt	83pt	68pt

Значение `\marginparsep` устанавливается таким же, как в «двусторонних» вариантах этих стилей, а значение `\evensidemargin` устанавливается таким же, как и `\oddsidemargin`.

Что же касается стиля `book`, то он, как известно, всегда является «двусторонним». Значения соответствующих параметров в нем таковы:

	Кегль		
	10pt	11pt	12pt
<code>\oddsidemargin</code>	0.5in	0.25in	0.25in
<code>\evensidemargin</code>	1.5in	1.25in	1.25in
<code>\marginparwidth</code>	0.75in	1in	1in
<code>\marginparsep</code>	7pt	7pt	7pt

Ширина страницы (`\textwidth`, как мы помним) принимает в стандартных стилях следующие значения:

Кегль	10pt	11pt	12pt
<code>article</code>	345pt	360pt	390pt
<code>report</code>	345pt	360pt	390pt
<code>book</code>	4.5in	5in	5in

В следующей таблице указаны значения параметров, относящихся к вертикальному размещению текста, как они определяются в стилях `article` и `report`. Параметр `\footskip` задает расстояние от низа основной части текста до нижнего колонтитула.

	Кегль		
	10pt	11pt	12pt
<code>\topmargin</code>	27pt	27pt	27pt
<code>\headheight</code>	12pt	12pt	12pt
<code>\headsep</code>	25pt	25pt	25pt
<code>\topskip</code>	10pt	10pt	10pt
<code>\baselineskip</code>	12pt	13.6pt	15pt
<code>\textheight</code>	526pt	526.8pt	550pt
<code>\footskip</code>	30pt	30pt	10pt

Не следует забывать, что значения `\baselineskip`, указанные в таблице, относятся только к шрифтам нормального размера. Значения `\textheight` подобраны таким образом, чтобы выполнялось условие со стр. 101.

Стиль `book` и тут идет своим путем. `\baselineskip` в нем такой же, как и в двух других стандартных стилях, `\headheight` также, как и в других стандартных стилях, всегда равен 12 пунктам, а вот остальные параметры таковы:

	Кегль		
	10pt	11pt	12pt
<code>\topmargin</code>	0.75in	0.73in	27pt
<code>\headsep</code>	0.25in	0.275in	0.275in
<code>\topskip</code>	10pt	10pt	10pt
<code>\textheight</code>	502pt	526.8pt	550pt
<code>\footskip</code>	0.35in	0.38in	30pt

2. Счетчики

Теперь перейдем к счетчикам, определяемым в стандартных стилях. Самые главные из них — это, конечно, те, что используются при нумерации разделов. В стиле `article` есть строчка

```
\newcounter{section}
```

в то время как в стилях `report` и `book`, в которых существуют еще и главы, определяется счетчик `chapter` для глав, а счетчик `section` определяется как подчиненный счетчику `chapter`:

```
\newcounter{chapter}
\newcounter{section}[chapter]
```

Остальные счетчики номеров разделов определяются во всех трех стандартных стилях одинаково:

```
\newcounter{part}
\newcounter{subsection}[section]
\newcounter{subsubsection}[subsection]
\newcounter{paragraph}[subsubsection]
\newcounter{subparagraph}[paragraph]
```

Соответствующие этим счетчикам `the`-команды определены в стиле `article` 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 стилях `report` и `book`, кроме того, определена `the`-команда для счетчика `chapter` и по-другому определена `\thesection`:

```
\renewcommand{\thechapter}{\arabic{chapter}}
\renewcommand{\thesection}{%
{\arabic{chapter}.\arabic{section}}}
```

За нумерацию сносок отвечает счетчик `footnote`. В стиле `article` этот счетчик определяется как никому не подчиненный:

```
\newcounter{footnote}
```

В стилях же `report` и `book` этот счетчик подчинен счетчику `chapter`, так как в них присутствует еще и команда

```
\@addtoreset{footnote}{chapter}
```

(см. раздел IX.2 по поводу `\@addtoreset`). В таком же положении, как `footnote`, находится и отвечающий за нумерацию формул счетчик `equation`: в стиле `article` он определен как никому не подчиненный, а в стилях `report` и `book` он подчинен счетчику `chapter`. Однако же в стилях `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 стиле `article` они никому не подчинены, а в двух других стандартных стилях они подчинены счетчику `chapter` и соответствующие `the`-команды определены как

```
\renewcommand{\thefigure}%
{\thechapter.\arabic{figure}}
```

(и аналогично для `table`).

Осталось рассказать про счетчики, связанные с нумерованными перечнями. Как объяснялось в разделе 2.5 главы VII, эти счетчики называются `enumi`, ..., `enumiv` в зависимости от уровня вложенности `enumerate`. Все эти счетчики, естественно, последовательно подчинены друг другу, а соответствующие `the`-команды и ссылочные префиксы (см. раздел 2 главы IX по поводу того, что такое ссылочный префикс) определены так (одинаково во всех трех стандартных стилях):

```
\renewcommand{\theenumi}{\arabic{enumi}}
\renewcommand{\theenumii}{\alph{enumii}}
\renewcommand{\p@enumii}{\theenumi}
\renewcommand{\theenumiii}{\roman{enumiii}}
\renewcommand{\p@enumiii}{\theenumi(\theenumii)}
\renewcommand{\theenumiv}{\Alph{enumiv}}
\renewcommand{\p@enumiv}{\p@enumiii\theenumiii}
```

Как объяснялось в разделе VII.2.5, оформление перечней зависит еще и от команд `\labelenumi`, ..., `\labelenumiv`. Они определены так (опять-таки одинаково во всех трех стилях):

```
\newcommand{\theenumi}{\arabic{enumi}}
\newcommand{\labelenumii}{(\theenumii)}
\newcommand{\labelenumiii}{\theenumiii.}
\newcommand{\labelenumiv}{\theenumiv.}
```

Коль скоро мы говорим про `enumerate`, скажем и про заголовки разделов перечня типа `itemize`. Они в стандартных стилях определены так:

```
\newcommand{\labelitemi}{\bullet}
\newcommand{\labelitemii}{\bf --}
\newcommand{\labelitemiii}{\ast}
\newcommand{\labelitemiv}{\cdot}
```

На самом деле заголовки для `itemize` определяются чуть хитрее. Например, настоящее определение `\labelitemi` выглядит так:

```
\newcommand{\labelitemi}{\mathsurround=0pt
\bullet}
```

Сделано это затем, чтобы не сделать вокруг «горошины» дополнительного пробела в случае, если пользователь решит установить ненулевое значение параметра `\mathsurround`. При этом, так как формула образует группу, в дальнейшем предыдущее значение `\mathsurround` восстановится. Полезно иметь в виду этот прием, если Вы пользуетесь математическими символами в качестве полиграфических значков.

3. Разделы документа

Определения команды `\section` и команд для создания более мелких разделов документа во всех трех стандартных стилях совершенно идентичны. Перечислим их:

```
\newcommand{\section}{\@startsection
  {section}{1}{0pt}{-3.5ex plus -1ex minus -.2ex}%
  {2.3ex plus .2ex}{\Large\bf}}
\newcommand{\subsection}{\@startsection
  {subsection}{2}{0pt}{-3.25ex plus -1ex minus-.2ex}%
  {1.5ex plus .2ex}{\large\bf}}
\newcommand{\subsubsection}{\@startsection
  {subsubsection}{3}{0pt}{-3.25ex plus -1ex minus-.2ex}%
  {1.5ex plus .2ex}{\normalsize\bf}}
\newcommand{\paragraph}{\@startsection
  {paragraph}{4}{0pt}{3.25ex plus 1ex minus .2ex}%
  {-1em}{\normalsize\bf}}
\newcommand{\subparagraph}{\@startsection
  {subparagraph}{4}{\parindent}{3.25ex plus 1ex minus
  .2ex}{-1em}{\normalsize\bf}}
```

Смысл аргументов команды `\@startsection` разъяснен в разделе 3.2 главы IX.

Теперь перечислим команды для создания оглавления, как они определены в стандартных стилях `book` и `report`:

```
\newcommand{\l@section}{\@dottedtocline
  {1}{1.5em}{2.3em}}
\newcommand{\l@subsection}{\@dottedtocline
  {2}{3.8em}{3.2em}}
\newcommand{\l@subsubsection}{\@dottedtocline
  {3}{7.0em}{4.1em}}
\newcommand{\l@paragraph}{\@dottedtocline
  {4}{10em}{5em}}
\newcommand{\l@subparagraph}{\@dottedtocline
  {5}{12em}{6em}}
```

В стиле `article` соответствующие команды определены несколько по-иному:

```
\newcommand{\l@subsection}{\@dottedtocline
  {2}{1.5em}{2.3em}}
\newcommand{\l@subsubsection}{\@dottedtocline
  {3}{3.8em}{3.2em}}
\newcommand{\l@paragraph}{\@dottedtocline
```



```
{4}{7.0em}{4.1em}}
\newcommand{\l@subparagraph}{\@dottedtocline
{5}{10em}{5em}}
```

Команда `\l@figure`, определяющая вид записи в списке иллюстраций, определена так (совершенно одинаково во всех трех стилях):

```
\newcommand{\l@figure}{\@dottedtocline{1}{1.5em}{2.3em}}
```

Точно так же выглядит и определение команды `\l@table`, отвечающей за вид записей в списке таблиц.

Команды, регулирующие оформление записей в оглавлении, во всех стандартных стилях определены одинаково:

```
\newcommand{\@pnumwidth}{1.55em}
\newcommand{\@tocrmarg}{2.55em}
\newcommand{\@dotsep}{4.5}
```

Осталось сказать про команды, определяющие записи в оглавлении для самых крупных подразделений документа. В стилях `report` и `book` это `\l@chapter`, а в стиле `article` это `\l@section`. Определение команды `\l@chapter` в стиле `book` мы привели в разделе IX.4; точно так же определяется эта команда в стиле `report`, и практически так же — команда `\l@section` в стиле `article`.

4. Колонтитулы

Теперь перейдем к командам, отвечающим за колонтитулы. Определение этих команд зависит не только от основного стиля, но и от стилевых опций с одной стороны, и от того, что сказано в команде `\pagestyle` — с другой. Предположим сначала, что основной стиль — `article`. В этом случае по умолчанию принято, что верхних колонтитулов на страницах нет, а нижние представляют из себя просто центрированные номера страниц. Если, однако, задана команда `\pagestyle` с аргументом `headings` или `myheadings`, то колонтитулы устроены более сложно. Нижних колонтитулов нет, точнее говоря, команды `\@oddfoot` и `\@evenfoot` определены как «ничего не делать»:

```
\newcommand{\@oddfoot}{}
\newcommand{\@evenfoot}{}

```

Что же до верхних колонтитулов, то их вид зависит от того, задана ли стилевая опция `twoside`. Для случая, когда она задана, соответствующие определения приведены в разделе 6 главы IX. Если же опция `twoside` не задана, то есть оформление «одностороннее», то верхние колонтитулы определяются так:

```
\newcommand{\@oddhead}{\sl \rightmark\hfil \rm\thepage}
```

(напомним, что при «одностороннем» стиле оформления документа команда `\@oddhead` задает колонтитулы как для четных, так и для нечетных страниц, а содержание команды `\@evenhead` при оформлении страницы игнорируется).

Наряду с командами для оформления колонтитулов, необходимо знать, каким образом команды, задающие разделы документа, автоматически ставят в текст пометки. Если аргументом команды `\pagestyle` был `myheadings`, то ответ прост: никак, и пользователь может вставлять пометки самостоятельно, если пожелает. Если же аргументом `\pagestyle`

был `headings`, то ответ опять-так зависит от того, является ли стиль оформления документа «двусторонним». Если является, то соответствующие определения уже были приведены в разделе 6 главы IX. К тому, что сказано по этому поводу в главе IX, следует только добавить, что в случае, когда секции (соотв. подсекции) не нумеруются из-за того, что значение счетчика `secnumdepth` меньше единицы (соотв. двух), то в пометки номер секции (подсекции) не включается. Если же стилевая опция `twoside` задана не была, то команда `\sectionmark` определяется так:

```
\newcommand{\sectionmark}[1]{\markright
{\uppercase{\thesection\hspace{1em}#1}}}
```

в то время как все остальные команды, создающие разделы документа, никаких пометок в текст не вносят. По-прежнему, если значение `secnumdepth` меньше единицы, то в этом определении следует опустить `\thesection\hspace{1em}`.

Теперь рассмотрим случай, когда основной стиль — `report`. В этом случае оформление с нетривиальными верхними и пустыми нижними колонтитулами принято по умолчанию. При этом команды `\@oddfoot` и `\@evenfoot` определены как «ничего не делать» (как и следовало ожидать). Команды для верхних колонтитулов определены точно так же, как в стиле `article` при наличии команды `\pagestyle{headings}`: Если стиль оформления документа «двусторонний» (то есть задана стилевая опция `twoside`), то так

```
\newcommand{\@evenhead}%
{\rm \thepage\hfil \sl \leftmark}
\newcommand{\@oddhead}%
{{\sl \rightmark}\hfil \rm\thepage}
```

(см. раздел IX.6), а если стиль «односторонний», то так:

```
\newcommand{\@oddhead}{{\sl \rightmark}\hfil \rm\thepage}
```

Различие со стилем `article` проявляется в том, какая информация и как передается для колонтитулов (если у нас была команда `\pagestyle` с параметром `headings`, то, как обычно, никакие пометки автоматически в текст не вносятся). Если стиль «двусторонний», то делается это следующим образом:

```
\newcommand{\chaptermark}[1]{\markboth
{\uppercase{\@charapp\ \thechapter. \ #1}}% левая пометка
}% правая пометка
}% конец макроопределения
\newcommand{\sectionmark}[1]{\markright
{\uppercase{\thesection. \ #1}}%
}% конец макроопределения
```

Напомним, что `\@charapp` означает то же, что и `\chaptername`, если не было команды `\appendix`, а после этой команды начинает означать то же, что и `\appendixname` — иными словами, это слово «Глава», «Приложение», и т. п. Никакие другие команды для создания разделов документа в этом случае пометок не вносят. Заметим еще, что, если значение счетчика `secnumdepth` таково, что главы (секции) не нумеруются, то в соответствующих аргументах команд `\markboth` и `\markright` будут отсутствовать номера глав (секций), а также слово «Глава» или его заменяющее (то есть команда `\@charapp`). Если же стиль «односторонний», то пометки вносит только команда `\chapter`:

```
\newcommand{\chaptermark}[1]{\markright
{\uppercase{\@chapapp\ \thechapter. \ #1}}
```

Осталось рассмотреть случай, когда основной стиль — `book`. По сравнению с уже сказанным, нам остается добавить совсем немного. В стиле `book` документ *всегда* оформляется «двусторонним» образом. Определения команд типа `\@oddhead`, `\chaptermark` и им подобных при этом дословно совпадают с этими определениями в «двустороннем» варианте стиля `report`.

5. Перечни

Как объяснялось в главе IX, для начала в стандартных стилях определяются параметры от `\leftmargini` до `\leftmarginvi` и им присваиваются определенные значения. Эти значения таковы:

	Кегль		
	10	11	12
<code>\leftmargini</code>	25pt	2.5em	2.5em
<code>\leftmarginii</code>	22pt	2.2em	2.2em
<code>\leftmarginiii</code>	18.7pt	1.87em	1.87em
<code>\leftmarginiv</code>	17pt	1.7em	1.7em
<code>\leftmarginv</code>	10pt	1em	1em
<code>\leftmarginvi</code>	10pt	1em	1em

После этого в стандартных стилях значение `\labelsep` устанавливается в 5 пунктов для кегля 10, и в 0.5em для кеглей 11 и 12; значение `\labelwidth` устанавливается равным разности `\leftmargini` и `\labelsep`.

Далее определяются команды `\@listI`, `\@listii` и т. п. Вот как звучат их определения для кегля 10:

```
\newcommand{\@listI}{\leftmargin=\leftmargini
\setlength{\parsep}{4pt plus 2pt minus 1pt}%
\setlength{\topsep}{8pt plus 2pt minus 4pt}%
\setlength{\itemsep}{4pt plus 2pt minus 1pt}}
```

(установка `\labelsep` и `\labelwidth` в этой команде не предусматривается, поскольку эти значения уже были установлены ранее, и они будут восстанавливаться всякий раз по выходе из окружений).

А вот команда `\@listii`:

```
\newcommand{\@listii}{\leftmargin=\leftmarginii
\labelwidth=\leftmarginii
\addtolength{\labelwidth}{-\labelsep}%
\setlength{\topsep}{4pt plus 2pt minus 1pt}%
\setlength{\parsep}{2pt plus 1pt minus 1pt}%
\itemsep=\parsep}
```

Обратите внимание, что команда `\@listii` не устанавливает нового значения параметра `\itemsep`, а пользуется тем же, что и на первом уровне вложенности: на второй уровень перечней

можно попасть только с первого, а на первом это значение будет уже установлено командой `\@listi`.

Теперь `\@listiii`:

```
\newcommand{\@listiii}{\leftmargin=\leftmarginiii
\labelwidth=\leftmarginiii
\addtolength{\labelwidth}{-\labelsep}%
\setlength{\topsep}{2pt plus 1pt minus 1pt}%
\setlength{\parsep}{0pt}%
\setlength{\partopsep}{0pt plus 0pt minus 1pt}%
\itemsep=\parsep}
```

Как видите, на третьем уровне меняется `\partopsep`. Далее:

```
\newcommand{\@listiv}{\leftmargin=\leftmarginiv
\labelwidth=\leftmarginiv
\addtolength{\labelwidth}{-\labelsep}}
```

На четвертом уровне, как видите, меняются только `\leftmargin` и, соответственно, `\labelwidth`, а остальные параметры переносятся в неизменности с предыдущего уровня. Определений для двух последних `\@list...`-команд мы не будем даже и приводить: они дословно совпадают с определением `\@listiv`, с тем естественным исключением, что в них ссылаются на `\leftmarginv` или `\leftmarginvi` вместо `\leftmarginiv`.

Все сказанное относилось к кеглю 10. Если кегль равен 11 или 12, то эти определения выглядят почти так же. Отличия проявляются только в том, какие значения присваивают параметрам команды `\@listI`, `\@listii` и (в кегле 12) `\@listiii`. Поэтому мы не будем выписывать макроопределений полностью, а только укажем соответствующие значения. Для кегля 11, в частности, они таковы:

	<code>\@listI</code>	<code>\@listii</code>
<code>\parsep</code>	4.5pt plus 2pt minus 1pt	2pt plus 1pt minus 1pt
<code>\topsep</code>	9pt plus 3pt minus 5pt	4.5pt plus 2pt minus 1pt
<code>\itemsep</code>	4.5pt plus 2pt minus 1pt	<code>\parsep</code>

Для кегля 12 эти значения таковы:

	<code>\@listI</code>	<code>\@listii</code>
<code>\parsep</code>	5pt plus 2.5pt minus 1pt	2.5pt plus 1pt minus 1pt
<code>\topsep</code>	10pt plus 4pt minus 6pt	5pt plus 2.5pt minus 1pt
<code>\itemsep</code>	5pt plus 2.5pt minus 1pt	<code>\parsep</code>

Наконец, команда `\@listiii` для кегля 12 отличается лишь тем, что присваивает параметру `\topsep` значение `2.5pt plus 1pt minus 1pt`.

Литература

- [1] L. Lamport: *Л_AT_EX, A Document Preparation System, User's Guide and Reference Manual*, Addison-Wesley Publishing Company (1985), ISBN 0-201-15790-X.
- [2] D. E. Knuth: *The T_EXbook*, часть А серии *Computers and Typesetting*, Addison-Wesley Publishing Company (1984), ISBN 0-201-13448-9. Русский перевод: Дональд Е. Кнут. Все про T_EX. М., 1993.
- [3] Н. Partl, I. Hyna. Л_AT_EX-Kurzbeschreibung.
- [4] Л_AT_EX: краткое описание.
- [5] М. Spivak. The Joy of T_EX. Русский перевод:...

Предметный указатель

- AMS-L^AT_EX, 7
- AMS-T_EX, 7
- listI), 226
- listii), 226
- ?', 63
- !', 63
- \+, 125
- \,, 51, 59
- \', 63
 - в окружении tabbing, 124
- \=, 63
- \[, 43
- \], 43
- \', 63
 - в окружении tabbing, 124
- \(, 43
- \), 43
- \{, 20, 40
- \|, 35
- \}, 20, 40
- \~, 63
- \!, 51
- \", 63
- *, 74
- \\
 - в абзаце, 73
 - в окружении tabbing, 121
 - в окружении tabular, 126
- \^, 63
- \ (backslash с пробелом), 61
- \-, 71
 - в окружении tabbing, 125
 - для предотвращения переносов, 73
- \., 63
- \/, 65
- \:, 51
- \;, 51
- \<, 125
- \=
 - в окружении tabbing, 121
- \>, 121
- \@, 61
- \@addtoreset, 180
- \@afterindentfalse, 186
- \@afterindenttrue, 186
- \@begintheorem, 204
- \@biblabel, 206
- \@chapapp, 185
- \@cite, 206
- \@dotsep, 189
- \@dottedtocline, 189
- \@eqnnum, 205
- \@evenfoot, 197
- \@evenhead, 197
- \@idxitem, 207
- \@listI, 193
- \@listii, 194
- \@listiii, 194, 226
- \@listiv, 194, 226
- \@listv, 194, 226
- \@listvi, 194, 226
- \@makechapterhead, 185
- \@makefnmark, 206
- \@makefntext, 206
- \@makeschapterhead, 186
- \@oddfont, 197
- \@oddhead, 197
- \@opargbegintheorem, 204
- \@pnumwidth, 189
- \@startsection, 182
- \@tocrmarg, 189
- \AA, 63
- \AE, 63
- \Alph, 145
- \Biggl, 41
- \Biggr, 41
- \Bigl, 41
- \Bigr, 41
- \Box, 35
- \Delta, 30

- \backslash Diamond, 35
 \backslash Downarrow, 32
 \backslash Gamma, 30
 \backslash H, 63
 \backslash Huge, 66
 \backslash Im, 35
 \backslash Join, 32
 \backslash L, 63
 \backslash LARGE, 66
 \backslash Lambda, 31
 \backslash Large, 66
 \backslash Leftarrow, 32
 \backslash Leftrightarrow, 32
 \backslash Longleftarrow, 32
 \backslash Longleftrightarrow, 32
 \backslash Longrightarrow, 32
 \backslash O, 63
 \backslash OE, 63
 \backslash Omega, 31
 \backslash P, 35
 \backslash Phi, 31
 \backslash Pi, 31
 \backslash Pr, 34
 \backslash Psi, 31
 \backslash Ralph, 214
 \backslash Re, 35
 \backslash Rightarrow, 32
 \backslash Roman, 145
 \backslash S, 35, 59
 \backslash Sigma, 31
 \backslash Theta, 30
 \backslash Uparrow, 32
 \backslash Updownarrow, 32
 \backslash Upsilon, 31
 \backslash Vert, 35
 \backslash Xi, 31
 \backslash aa, 63
 \backslash abstractname, 104
 \backslash acute, 42
 \backslash addcontentsline, 187
 \backslash addtocontents, 187
 \backslash addtocounter, 144
 \backslash addtolength, 155
 \backslash ae, 63
 \backslash afterthmseparator, 214
 \backslash aleph, 34
 \backslash alph, 145
 \backslash alpha, 30
 \backslash amalg, 31
 \backslash angle, 34
 \backslash appendix, 104
 \backslash appendixname, 104
 \backslash approx, 31
 \backslash arabic, 144
 \backslash arcctg, 213
 \backslash arctg, 213
 \backslash arraycolsep, 133
 \backslash arrayrulewidth, 133
 \backslash arraystretch, 135
 \backslash ast, 35
 \backslash asymp, 32
 \backslash atop, 44
 \backslash author, 105
 \backslash b, 63
 \backslash backslash, 35, 40
 \backslash bar, 42
 \backslash baselineskip, 101, 220
 \backslash baselinestretch, 81
 \backslash batchmode, 28
 \backslash beta, 30
 \backslash bf, 13, 66
 в формулах, 37
 \backslash bibitem, 107
 с необязательным аргументом, 107
 \backslash bibname, 104
 \backslash bigcap, 34
 \backslash bigcirc, 31
 \backslash bigcup, 34
 \backslash biggl, 41
 \backslash biggr, 41
 \backslash bigl, 41
 \backslash bigodot, 34
 \backslash bigoplus, 34
 \backslash bigotimes, 34
 \backslash bigrr, 41
 \backslash bigskip, 79
 \backslash bigskipamount, 74
 \backslash bigsqcup, 34
 \backslash bigtriangledown, 31
 \backslash bigtriangleup, 31
 \backslash biguplus, 34
 \backslash bigvee, 34
 \backslash bigwedge, 34
 \backslash boldmath, 38
 \backslash bot, 35
 \backslash bowtie, 32

- `\breve`, 42
- `\bullet`, 31
- `\с`, 63
- `\cal`, 38
- `\cap`, 31
- `\caption`, 112
 - с необязательным аргументом, 113
- `\cdot`, 31
- `\cdots`, 21
- `\centerline`, 85
- `\ch`, 213
- `\chapter`, 103
- `\chaptername`, 104
- `\check`, 42
- `\chi`, 30
- `\choose`, 44
- `\circ`, 31
- `\circle`, 117
- `\circle*`, 117
- `\cite`, 107
 - с необязательным аргументом, 107
- `\cleardoublepage`, 82
- `\clearpage`, 82
- `\cline`, 128
- `\clubsuit`, 35
- `\colon`, 54
- `\columnsep`, 101
- `\columnseprule`, 101
- `\cong`, 31
- `\contentsname`, 104, 191
- `\coprod`, 34
- `\copy`, 176
- `\copyright`, 35, 59
- `\cosec`, 213
- `\countpages`, 215
- `\ctg`, 213
- `\cth`, 213
- `\cup`, 31
- `\d`, 63
- `\dag`, 35
- `\dagger`, 31
- `\dashv`, 32
- `\date`, 105
- `\ddag`, 35
- `\ddagger`, 31
- `\ddot`, 42
- `\ddots`, 46
- `\delta`, 30
- `\det`, 34
- `\diamond`, 31
- `\diamondsuit`, 35
- `\displaystyle`, 52
- `\div`, 31
- `\documentstyle`, 12, 178
- `\dontcountpages`, 215
- `\dot`, 42
- `\doteq`, 31
- `\dotfill`, 169
- `\dots`, 59
- `\downarrow`, 32
- `\ell`, 35
- `\em`, 65
- `\emergencystretch`, 72, 76
- `\emptyset`, 34
- `\endinput`, 23
- `\enskip`, 62
- `\eps`, 213
- `\epsilon`, 30
- `\eqno`, 36
- `\equiv`, 31
- `\eta`, 30
- `\evensidemargin`, 101, 219, 220
- `\exhyphenpenalty`, 77
- `\exists`, 34
- `\fbox`, 60
- `\fboxrule`, 164
- `\fboxsep`, 164
- `\figurename`, 104, 112
- `\flat`, 35
- `\flushbottom`, 84
- `\fnsymbol`, 145
- `\footnote`, 67
- `\footnotemark`, 68
- `\footnoterule`, 205
- `\footnotesep`, 206
- `\footnotesize`, 66
- `\footnotetext`, 68
- `\footskip`, 197, 220
- `\forall`, 34
- `\frac`, 20
- `\framebox`, 164
- `\frenchspacing`, 62
- `\frown`, 32
- `\fussy`, 72
- `\gamma`, 30
- `\gcd`, 34

- `\ge`, 19, 31
- `\geq`, 35
- `\gets`, 32
- `\gg`, 31
- `\glqq`, 216
- `\grave`, 42
- `\grqq`, 216
- `\halign`, 136
- `\hangafter`, 91
- `\hangindent`, 91
- `\hat`, 42
- `\hbar`, 34
- `\hbox`, 166
- `\headheight`, 101, 198, 220
- `\headsep`, 101, 197, 198, 220
- `\heartsuit`, 35
- `\hfil`, 168
- `\hfill`, 168
- `\hfuzz`, 75
- `\hline`, 127
- `\hoffset`, 102
- `\hookleftarrow`, 32
- `\hookrightarrow`, 32
- `\hphantom`, 54
- `\hrule`, 94
- `\hrulefill`, 169
- `\hspace`, 62
- `\hspace*`, 62
- `\hss`, 171
- `\huge`, 66
- `\hyphenation`, 71
- `\hyphenpenalty`, 77
- `\i`, 63
- `\imath`, 35
- `\in`, 31
- `\index`, 109
- `\indexname`, 104
- `\indexspace`, 108
- `\inf`, 34
- `\infty`, 34
- `\input`, 22
- `\int`, 34
- `\iota`, 30
- `\it`, 66
- `\item`, 86
 - в окружении `theindex`, 108
 - квадратная скобка после команды, 88
 - необязательный аргумент, 87, 89
- `\itemsep`, 193, 227
- `\j`, 63
- `\jmath`, 35
- `\kappa`, 30
- `\kill`, 122
- `\l`, 63
- `\label`, 17
- `\labelenumi`, 153, 222
- `\labelenumii`, 153, 222
- `\labelenumiii`, 153, 222
- `\labelenumiv`, 153, 222
- `\labelitemi`, 152, 222
- `\labelitemii`, 153
- `\labelitemiii`, 153
- `\labelitemiv`, 153
- `\labelrlist`, 216
- `\labelsep`, 192, 226
- `\labelwidth`, 192, 226
- `\lambda`, 30
- `\land`, 35
- `\langle`, 40
- `\large`, 66
- `\lbrace`, 35
- `\lbrack`, 35
- `\lceil`, 40
- `\ldotp`, 54
- `\ldots`, 21, 59
- `\le`, 19, 31
- `\leaders`, 169, 177
- `\leadsto`, 32
- `\left`, 20, 39
- `\leftarrow`, 35
- `\lefteqn`, 48, 50, 54, 172
- `\leftharpoondown`, 32
- `\leftharpoonup`, 32
- `\leftmargin`, 192
- `\leftmargini`, 194, 226
- `\leftmarginii`, 194, 226
- `\leftmarginiii`, 226
- `\leftmarginiv`, 226
- `\leftmarginv`, 226
- `\leftmarginvi`, 194, 226
- `\leftmark`, 199
- `\leftrightarrow`, 32
- `\leftskip`, 190
- `\leq`, 35
- `\leqno`, 36
- `\lfloor`, 40

- `\lim`, 34
- `\liminf`, 34
- `\limits`, 34
- `\limsup`, 34
- `\line`, 117
- `\linebreak`, 73
 - с необязательным аргументом, 74
- `\linethickness`, 120
- `\listfigurename`, 104
- `\listoffigures`, 113
- `\listoftables`, 113
- `\listparindent`, 193
- `\listtablename`, 104
- `\ll`, 31
- `\llap`, 116
- `\lnot`, 35
- `\longleftarrow`, 32
- `\longleftrightarrow`, 32
- `\longmapsto`, 32
- `\longrightarrow`, 32
- `\looseness`, 77
- `\lor`, 35
- `\lowercase`, 200
- `\magstep`, 210
- `\magstephalf`, 210
- `\makebox`, 161
- `\makeindex`, 109
- `\maketitle`, 105
- `\mapsto`, 32
- `\marginpar`, 113
 - с необязательным аргументом, 114
- `\marginparpush`, 114
- `\marginparsep`, 114, 219, 220
- `\marginparwidth`, 114, 219, 220
- `\markboth`, 199
- `\markright`, 200
- `\mathbin`, 55
- `\mathop`, 55
- `\mathrel`, 55
- `\mathstrut`, 53
- `\mathsurround`, 57, 206, 223
- `\max`, 34
- `\mbox`, 73, 161
 - в формулах, 38
 - для предотвращения переноса, 73
 - как «пустой текст» на странице, 82
- `\medskip`, 79
- `\medskipamount`, 74
- `\mho`, 35
- `\mid`, 32
- `\min`, 34
- `\mit`, 37
- `\models`, 32
- `\mp`, 31
- `\mu`, 30
- `\multicolumn`, 128
- `\multirow`, 118, 176
- `\nabla`, 34
- `\natural`, 35
- `\ne`, 31
- `\nearrow`, 32
- `\neg`, 34
- `\neq`, 35
- `\newcommand`, 137
- `\newcounter`, 144
 - с необязательным аргументом, 147
- `\newenvironment`, 156
- `\newfont`, 209
- `\newlength`, 154
- `\newpage`, 82
- `\newtheorem`, 158
- `\ni`, 31
- `\noindent`, 79
- `\nolimits`, 34
- `\nolinebreak`, 74
- `\nonfrenchspacing`, 62
- `\nonstopmode`, 28
- `\nonumber`, 47
- `\nopagebreak`, 82
- `\normalmarginpar`, 114
- `\normalsize`, 66
- `\not`, 41
- `\notin`, 31
- `\nu`, 30
- `\nrightarrow`, 32
- `\o`, 63
- `\oddsidemargin`, 100, 219, 220
- `\odot`, 31
- `\oe`, 63
- `\oint`, 34
- `\omega`, 30
- `\ominus`, 31
- `\onecolumn`, 83
- `\oo`, 216
- `\oplus`, 31
- `\oslash`, 31

- `\otimes`, 31
- `\oval`, 118
 - с необязательным аргументом, 118
- `\overbrace`, 45
- `\overleftarrow`, 43
- `\overline`, 42
- `\overrightarrow`, 43
- `\owns`, 35
- `\pagebreak`, 83
- `\pagenumbering`, 99
- `\pageref`, 17
- `\pagestyle`, 99
- `\par`, 78
- `\paragraph`, 103
- `\parallel`, 31
- `\parbox`, 163
 - с необязательным аргументом, 163
- `\parindent`, 14
- `\parsep`, 193, 227
- `\parshape`, 93
- `\parskip`, 81
- `\part`, 103
- `\partial`, 34
- `\partname`, 104
- `\partopsep`, 193
- `\perp`, 31
- `\phantom`, 53
- `\phi`, 30
- `\pi`, 30
- `\pm`, 31
- `\poptabs`, 124
- `\postchapter`, 214
- `\postsection`, 214
- `\pounds`, 35, 59
- `\prec`, 32
- `\preceq`, 32
- `\presection`, 214
- `\presubsection`, 214
- `\prime`, 34
- `\prod`, 34
- `\propto`, 32
- `\protect`, 111, 187
 - в аргументе `\addtocontents`, 187
 - в аргументе `\addcontentsline`, 188
 - в пометке, 202
- `\psi`, 30
- `\pushtabs`, 124
- `\put`, 116
- `\qqquad`, 49, 51, 62
- `\quad`, 51, 62
- `\raggedbottom`, 84
- `\raggedright`, 74
- `\raisebox`, 165
- `\ralph`, 214
- `\rangle`, 40
- `\rbrace`, 35
- `\rbrack`, 35
- `\rceil`, 40
- `\ref`, 18
 - в окружении `enumerate`, 88
 - ссылка на плавающую иллюстрацию, 112
 - ссылка на счетчик, определенный пользователем, 148
 - ссылки на раздел документа, 102
 - ссылки на формулы, 36
- `\refname`, 104
- `\refstepcounter`, 147
- `\relax`, 95
- `\renewcommand`, 141
- `\renewenvironment`, 157
- `\reversemarginpar`, 114
- `\rfloor`, 40
- `\rho`, 30
- `\right`, 20, 39
- `\rightarrow`, 35
- `\rightharpoondown`, 32
- `\rightharpoonup`, 32
- `\rightthyphenmin`, 70
- `\rightleftharpoons`, 32
- `\rightmargin`, 192
- `\rightmark`, 199
- `\rightskip`, 190
- `\rlap`, 172
- `\rm`, 13, 66
 - в формулах, 37
- `\roman`, 145
- `\rule`, 93
 - с необязательным аргументом, 94
- `\samepage`, 82
- `\savebox`, 176
- `\sbox`, 176
- `\sc`, 66
- `\scriptscriptstyle`, 52
- `\scriptsize`, 66
- `\scriptstyle`, 52

- `\scrollmode`, 28
- `\searrow`, 32
- `\section`, 102
 - с необязательным аргументом, 102
- `\section*`, 103
- `\sectionmark`, 200
- `\setcounter`, 144
- `\setlength`, 154
- `\setminus`, 31
- `\sf`, 66
- `\sh`, 213
- `\sharp`, 35
- `\sigma`, 30
- `\sim`, 31
- `\simeq`, 31
- `\sl`, 12, 66
- `\sloppy`, 71
- `\small`, 66
- `\smallskip`, 79
- `\smallskipamount`, 74
- `\smash`, 54
- `\smile`, 32
- `\spadesuit`, 35
- `\special`, 115
- `\sqcap`, 31
- `\sqcup`, 31
- `\sqrt`, 21
- `\sqsubseteq`, 32
- `\sqsupseteq`, 32
- `\ss`, 63
- `\stackrel`, 44
- `\star`, 31
- `\stepcounter`, 148
- `\strut`, 95, 96, 135, 198
- `\subitem`, 108, 207
- `\subparagraph`, 103
- `\subsection`, 103
- `\subsectionmark`, 200
- `\subset`, 31
- `\subseteq`, 31
- `\subsubitem`, 108, 207
- `\subsubsection`, 103
- `\succ`, 32
- `\succeq`, 32
- `\sum`, 34
- `\sup`, 34
- `\supset`, 31
- `\supseteq`, 31
- `\surd`, 35
- `\swarrow`, 32
- `\symbol`, 60
- `\t`, 63
- `\tabcolsep`, 133
- `\tablename`, 104, 113
- `\tableofcontents`, 106
 - стандартное определение, 191
- `\tau`, 30
- `\textheight`, 101, 220
- `\textstyle`, 52
- `\textwidth`, 100, 220
- `\tg`, 213
- `\th`, 213
- `\thanks`, 105
- `\theindex`, 207
- `\theta`, 30
- `\thicklines`, 120
- `\thinlines`, 120
- `\thispagestyle`, 99
- `\tilde`, 42
- `\times`, 31
- `\tiny`, 66
- `\title`, 105
- `\to`, 32
- `\today`, 216
- `\tolerance`, 76
- `\top`, 35
- `\topmargin`, 101, 220
- `\topsep`, 193, 227
- `\topskip`, 101, 220
- `\triangle`, 34
- `\triangleleft`, 31
- `\triangleright`, 31
- `\tt`, 66
- `\twocolumn`, 83
- `\u`, 63
- `\uchyph`, 78
- `\underbrace`, 45
- `\underline`, 60
- `\unitlength`, 116
- `\uparrow`, 32
- `\updownarrow`, 32
- `\uplus`, 31
- `\uppercase`, 200
- `\upsilon`, 30
- `\usebox`, 176
- `\usecounter`, 195

- `\v`, 63
- `\value`, 145
- `\varepsilon`, 30
- `\varphi`, 30
- `\varpi`, 30
- `\varrho`, 30
- `\varsigma`, 30
- `\vartheta`, 30
- `\vbox`, 174
- `\vdash`, 32
- `\vdots`, 46
- `\vec`, 42
- `\vector`, 117
- `\vee`, 31
- `\verb`, 91
 - в сносках, 91
- `\verb*`, 91
- `\voffset`, 102
- `\vphantom`, 54
- `\vphi`, 213
- `\vrule`, 94
- `\vspace`, 80
- `\vspace*`, 80
- `\wedge`, 31
- `\widehat`, 42
- `\widetilde`, 43
- `\wp`, 35
- `\wr`, 31
- `\xi`, 30
- `\zeta`, 30
- `\лк`, 216
- `\пк`, 216
- `\сегодня`, 216
- TeXовское приглашение, 26
- 11pt (стилевая опция), 98
- 12pt (стилевая опция), 98
- abstract (окружение), 104
- array (окружение), 45, 125
- article (стиль), 97
- at (ключевое слово), 210
- aux-файл, 18
- book (стиль), 97
- center (окружение), 15, 85
- depth (ключевое слово), 95, 173
- description (окружение), 89
- displaymath (окружение), 43
- draft (стилевая опция), 98
- empty (стиль оформления страниц), 99
- enumerate (окружение), 88, 153
- enumiii (счетчик), 153, 222
- enumii (счетчик), 153, 222
- enumiv (счетчик), 153, 222
- enumi (счетчик), 153, 222
- eqnarray* (окружение), 48
- eqnarray (окружение), 47
- equation (окружение), 35
- figure* (окружение), 113
- figure (окружение), 112
 - с необязательным аргументом, 112
- figure (счетчик), 222
- fleqn (стилевая опция), 98
- flushleft (окружение), 85
- flushright (окружение), 85
- footnote (счетчик), 221
- headings (стиль оформления страниц), 99, 224
- height (ключевое слово), 95
- idx-файл, 109
- itemize (окружение), 87, 152
- l@-команда, 188
- leqno (стилевая опция), 98
- list (окружение), 195
- lof-файл, 113, 187
- log-файл, 23
- lot-файл, 113, 187
- math (окружение), 43
- minipage (окружение), 164
- myheadings (стиль оформления страниц), 202, 224
- picture (окружение), 115
 - вложенные окружения, 119
- pk-файлы, 209
- plain (стиль оформления страниц), 99
- quotation (окружение), 84
- quote (окружение), 84, 196
- report (стиль), 97
- rlist (окружение), 215
- scaled (ключевое слово), 210
- secnumdepth (счетчик), 181
- tabbing (окружение), 121
- table (окружение), 113
- table (счетчик), 222
- tabular (окружение), 125
- tfm-файлы, 209
- thebibliography (окружение), 106
- theindex (окружение), 108

- titlepage (окружение), 106
titlepage (стилевая опция), 98, 105
tocdepth (счетчик), 181
toc-файл, 106
totalpages (счетчик), 215
to (ключевое слово), 167
trivlist (окружение), 196
twocolumn (стилевая опция), 98
twoside (стилевая опция), 98
verbatim* (окружение), 91
verbatim (окружение), 90
 в сносках, 91
verse (окружение), 86
width (ключевое слово), 95
- Badness, 75
- Overfull, 69
 в колонтитуле, 102
- Plain T_EX, 7
- Underfull, 70
 при нехватке клея, 167
 при печати страницы, 83
- Абзацы, 10, 68
 абзацный отступ, 14
 верстка без выравнивания, 74
 дополнительный интервал между абзацами, 81
 изменение количества строк, 77
 неправильной формы, 93
 нестандартной формы, 91
 неточное выравнивание по правому краю, 75
 подавление отступа, 79
 сообщения о трудностях при верстке, 69, 70
- Автор документа, 105
- Амперсенд, 126
- Аргумент, 15, 64
 необязательный, 15
 перемещаемый, 111
- Базисная линия (baseline), 160
- Бинарные операции, 31, 54, 55
- Бинарные отношения, 31, 32, 54, 55
- Биномиальные коэффициенты, 44
- Блок (box), 160
- Блочные переменные, 175
- Буквы греческие, 30, 31
 наклонные, 37
- Группы, 13
- Дефис, 58
- Диаграммы, 48
- Диакритические знаки, 63
 в окружении tabbing, 123
- Длина
 единицы измерения, 16
 em, 17
 ex, 17
 пика, 16
 пункт, 16
 параметры, 15, 154
 присваивание значений, 154
 с коэффициентом, 155
 сложение, 155
- Дроби, 20
- Жидкие строки, 69
 равномерное увеличение разреженности, 72
 снятие запретов, 71
- Заглавие документа, 105
- Заметки на полях, 113
- Иллюстрации, 112
 при наборе в две колонки, 113
 стандартное название, 112
- Индексы, 18, 19
- Интеграл, 34
 контурный, 34
- Интерлиньяж, 81
- Кавычки, 59
 елочки, 59
 лапки, 59
- Кернинг, 58
- Клей, 80, 170
 бесконечно сжимаемый, 171
- Кнут, Дональд, 7
- Колонтитулы, 197
 высота, 198
 интервал между колонтитулом и текстом, 197
 передача информации из текста, 199

- Команды, 11
пробел после имени, 12
символ @ в роли буквы, 179
со звездочкой, 16
хрупкие (fragile), 111
- Комментарии, 10
- Корень, 21
- Лампорт, Лесли, 7
- Лигатуры, 58
- Лидеры, 169
в оглавлении, 189
использование блоковых переменных, 177
- Линейки, 93
невидимые, 95, 132, 135
в формулах, 53
- Макросы, 137
новые окружения, 156
с аргументами, 142
- Масштабирование, 210
- Матрицы, 45
преамбула, 46
- Метафонт, 209
- Многоточие, 59
в тексте, 59
в формулах, 21
- Надстрочные знаки
в тексте, 63
в формулах, 42
- Неразрывный пробел ~, 60
- Оглавление, 106
запись текста без номера страницы, 187
запись текста с номером страницы, 187
модификация оформления, 186
стандартный заголовок, 106
степень детализации, 181
- Ограничители, 40
- Окружения, 15
типа «теорема», 158
- Операторы типа сумма, 34, 54, 55
- Опции стилевые, 15, 98
дополнительные стилевые файлы, 179
- Переносы
в словах с дефисом, 70
в словах, начинающихся с прописной буквы, 78
двух последних букв, 70
затруднение и запрет, 77, 78
предотвращение в данном слове
с помощью \mbox, 73
с помощью \-, 73
указание разрешенных мест
«глобальное», 71
«локальное», 71
- Перечеркнутые символы, 41
- Перечни
itemize, 87
модификация
заголовков enumerate, 153
заголовков itemize, 152
нумерованные (enumerate), 88
с заголовками (description), 89
- Плавающие иллюстрации
при наборе в две колонки, 113
стандартное название, 112
- Плавающие таблицы, 113
стандартное название, 113
- Подчеркивание, 60
- Поля, 100
верхнее и нижнее, 101
левое и правое, 100, 101
- Пометки (marks), 199
автоматическое внесение в текст, 200
- Преамбула документа, 13
- Предметный указатель, 108
модификация оформления, 207
стандартное заглавие, 109
- Промежутки
в формулах, 51, 55
вертикальные, 79, 80
горизонтальные, 62
дополнительные после конца предложения, 61
- Разделы
варианты со звездочкой, 103
заголовки, входящие в абзац, 184
модификация оформления, 182, 185, 186
подавление отступа в первом абзаце, 103, 183

- подавление отступа в первом абзаце главы, 186
стиль оформления заголовка, 183
уровень вложенности, 181
- Рамки, 60, 164
- Режимы TeXa, 78
- Скобки, 20
горизонтальные, 45
переменного размера, 20, 39–41
- Сноски, 67
к тексту внутри блока, 68
к титульному листу, 105
линейка, отделяющая сноски от текста, 205
оформление номеров, 206
оформление текста сноски, 206
пробел между страницей и сносками, 205
- Спивак, Майкл, 8
- Список иллюстраций, 113
- Список литературы, 106
скобки вокруг номера ссылки, 207
стандартное заглавие, 108
- Список таблиц, 113
стандартное заглавие, 104
- Сравнения по модулю, 33
- Степени, 18, 19
- Стилевой файл, 178
- Стихи, 86
- Страницы
запрет разрыва, 82
глобальный, 82
принудительный разрыв, 82
при двустороннем наборе, 82
с выдачей плавающих иллюстраций, 82
стиль оформления, 99
модификация, 197
- Стрелки, 32
буква над стрелкой, 45
надпись справа от стрелки, 48
- Строки
запрет разрыва, 74
неразрывный пробел, 60
насилованный разрыв, 73
с выравниванием, 73
- Счетчики, 143
- the-команда, 149
выдача на печать, 144, 145
определенные при начале трансляции, 152
переподчинение существующего счетчика, 180
подчинение, 147
присваивание значений, 144
сложение, 144
создание, 144
ссылочный префикс, 180
увеличение на шаг, 147
- Таблицы
абзац в графе, 129
вертикальные линейки, 127
горизонтальные линейки, 127
графы в несколько колонок, 128
интервал между колонами, 133
невидимые линейки в строках, 135
преамбула, 126
at-выражения, 133
символ |, 127
разбиение преамбулы на колонки, 129
толщина линеек, 133
частичные горизонтальные линейки, 128
- Табуляция, 121
выравнивание по правому краю, 124
запоминание и вспоминание позиций, 124
использование позиций, 121
предварительная установка позиций, 122
сдвиг первой позиции, 125
установка позиций, 121
- Тангенс, 32
- Тире, 58
длинное (em-dash), 58
короткое (en-dash), 58
- Титульный лист, 105
дополнительная информация, 105
оформление вручную, 106
сноски, 105
- Точка отсчета, 116, 160
- Формулы
альтернативные обозначения, 43
включение текста, 38

внутритекстовые, 18, 43
выключные, 18, 43, 98
 знак препинания после формулы, 21
надстрочные знаки, 42
нумерация, 35, 36, 98, 180, 205
переносы, 37, 49, 50

Шрифты

typewriter, 66, 212
в формулах, 37, 218
каллиграфический, 38
капитель, 66
коррекция наклона, 65
курсивный, 66, 211
математический курсив, 18
наклонный, 12, 66
новая схема выбора, 217
полужирный, 13, 66
прямой светлый (roman), 66, 211
рубленный, 66
Штрихи (в формулах), 21, 35, 56