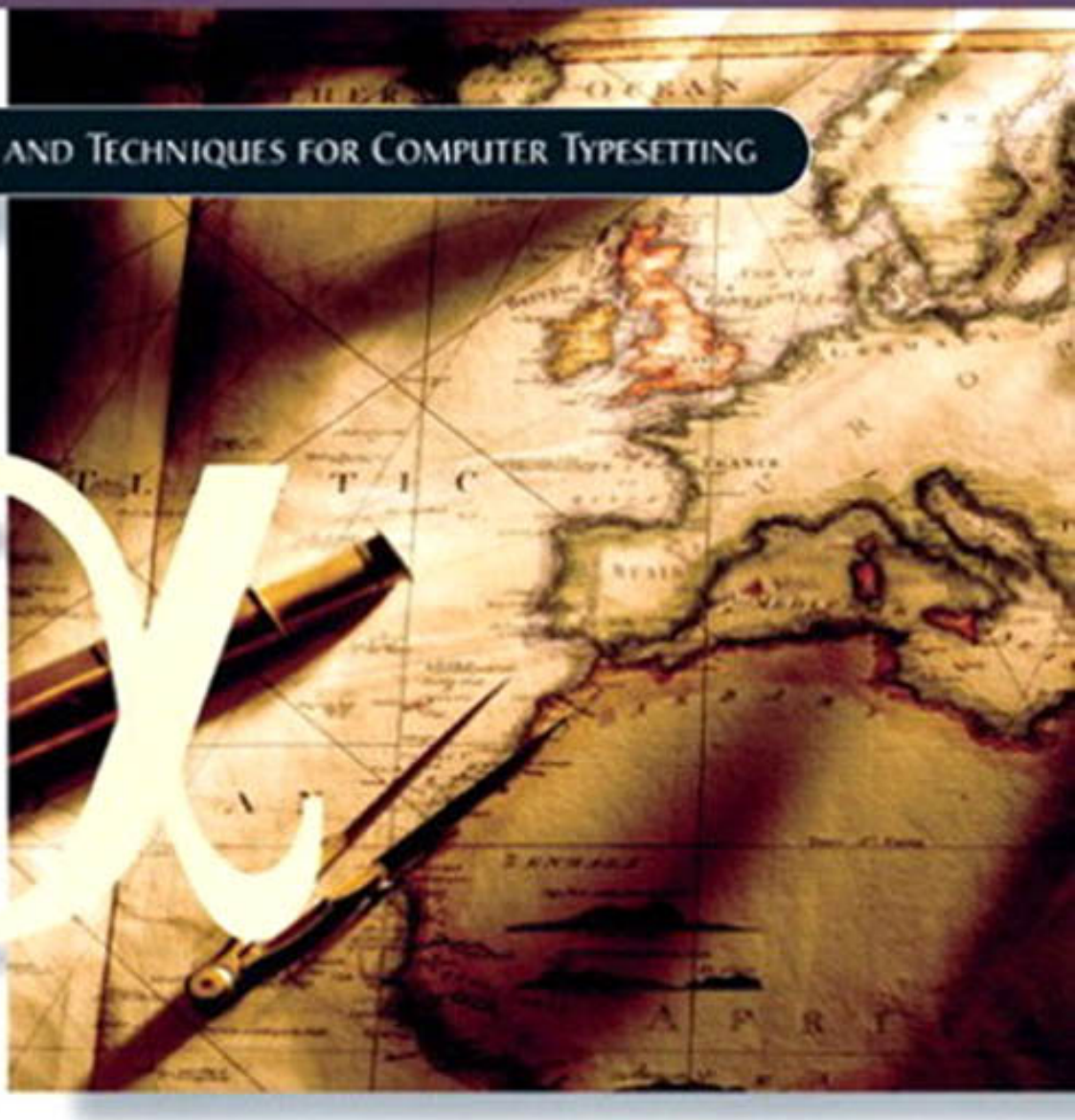
Guide to LaTeX

Fourth Edition

Tools and Techniques for Computer Typesetting



Helmut Kopka and Patrick W. Daly

Guide to LaTeX

Fourth Edition

Addison-Wesley Series on Tools and Techniques for Computer Typesetting

This series focuses on tools and techniques needed for computer typesetting and information processing with traditional and new media. Books in the series address the practical needs of both users and system developers. Initial titles comprise handy references for LaTeX users; forthcoming works will expand that core. Ultimately, the series will cover other typesetting and information processing systems, as well, especially insofar as those systems offer unique value to the scientific and technical community. The series goal is to enhance your ability to produce, maintain, manipulate, or reuse articles, papers, reports, proposals, books, and other documents with professional quality.

Ideas for this series should be directed to the editor: `mittelbach@aw.com`.
Send all other comments to the publisher: `awprofessional@aw.com`.

Series Titles

Guide to LaTeX, Fourth Edition, by Helmut Kopka and Patrick W. Daly

The LaTeX Companion, Second Edition, by Frank Mittelbach and Michel Goossens with Johannes Braams, David Carlisle, and Chris Rowley

The LaTeX Graphics Companion, Second Edition, by Michel Goossens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel, and Herbert Voß

The LaTeX Web Companion, by Michel Goossens and Sebastian Rahtz

Also from Addison-Wesley:

LaTeX: A Document Preparation System, Second Edition, by Leslie Lamport

The Unicode Standard, Version 5.0, by the Unicode Consortium

Guide to LaTeX

Fourth Edition

Helmut Kopka
Patrick W. Daly

Addison-Wesley
Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data
Kopka, Helmut.
Guide to LaTeX/Helmut Kopka, Patrick W. Daly.—4th ed.
p. cm.
Include bibliographical references and index.
ISBN 0-321-17385-6 (pbk. : alk. paper)
1. LaTeX (Computer file) 2. Computerized typesetting. I. Daly, Patrick W. II. Title
Z253.4.L38K66 2004
686.2'2544536—dc22 2003060364

ISBN 0-321-17385-6
Text printed in the United States on recycled paper at Courier Stoughton in Stoughton, Massachusetts.
5 6 7 8 9 10—CRS—0908070605
9th Printing February 2008

Trademark Notices

METAFONT™ is a trademark of Addison-Wesley Publishing Company.

$\mathcal{AMS}$-TEX™ and $\mathcal{AMS}$-LATEX™ are trademarks of the American Mathematical Society.

Lucida® is a registered trademark of Bigelow & Holmes.

Microsoft®, MS-DOS®, Windows®, and Internet Explorer® are registered trademarks of Microsoft Corporation.

PostScript®, Acrobat Reader®, and Acrobat logo® are registered trademarks, and PDF™ is a trademark of Adobe Systems Incorporated.

UNIX® is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Limited.

VAX™ and VMS™ are trademarks of Digital Equipment Corporation.

IBM® is a registered trademark of International Business Machines Corporation.

Netscape™ and Netscape Navigator™ are trademarks of Netscape Communications Corporation.

TrueType™ is a trademark, and Apple® and Macintosh® are registered trademarks, of Apple Computer Inc.

License Notice

This page intentionally left blank

Contents

Preface

A new edition to *A Guide to LaTeX* begs the fundamental question: Has LaTeX changed so much since the appearance of the third edition in 1999 that a new release of this book is justified?

The simple answer to that question is 'Well,. . . .' In 1994, the LaTeX world was in upheaval with the issue of the new version LaTeX 2ε, and the second edition of the *Guide* came out just then to act as the bridge between the old and new versions. By 1998, the initial teething problems had been worked out and corrected through semiannual releases, and the third edition could describe an established, working system. However, homage was still paid to the older 2.09 version since many users still employed its familiar syntax, although they were most likely to be using it in a LaTeX 2ε environment. LaTeX has now reached a degree of stability that since 2000 the regular updates have been reduced to annual events, which often appear months after the nominal date, something that does not worry anyone. The old version 2.09 is obsolete and should no longer play any role in such a book. In this fourth edition, it is reduced to an appendix just to document its syntax and usage.

But if LaTeX itself has not changed substantially since 1999, many of its peripherals have. The rise of programs such as pdfTeX and `dvipdfm` for PDF output adds new possibilities, which are realized, not in LaTeX directly, but by means of more modern *packages* to extend the basic features. The distribution of TeX/LaTeX installations has changed, such that most users are given a complete, ready-to-run setup, with all the 'extras' that previously had to be obtained separately. Those extras include user-contributed packages, many of which are now considered indispensable. Today 'the LaTeX system' includes much more than the basic kernel by Leslie Lamport, encompassing the contributions of hundreds of other people. This edition reflects this increase in breadth.

The changes to the fourth edition are mainly those of emphasis.

1. The material has been reorganized into 'Basics' and 'Beyond the Basics' ('advanced' sounds too intimidating) while the appendices contain topics that can be skipped by most everyday users. One exception: Appendix G is an alphabetized command summary that many people find extremely useful (including ourselves).

   This reorganizing is meant to stress certain aspects over others. For example, the section on graphics inclusion and color was originally treated as an exotic extra, relegated to an appendix on extensions; in the third edition, it was moved up to be included in a front chapter along with the `picture` environment and

floats; now it dominates Chapter 8 all on its own, the floats come in the following Chapter 9, and `picture` is banished to the later Chapter 16. This is not to say that the `picture` features are no good, but only that they are very specialized. We add descriptions of additional drawing possibilities there too.

2. It is stressed as much as possible that LaTeX is a *markup* language, with separation of content and form. Typographical settings should be placed in the preamble, while the body contains only logical markup. This is in keeping with the modern ideas of XML, where form and content are radically segregated.

3. Throughout this edition, contributed packages are explained at the point in the text where they are most relevant. The `fancyhdr` package comes in the section on page styles, `natbib` where literature citations are explained. This stresses that these 'extensions' are part of the LaTeX system as a whole. However, to remind users that they must still be explicitly loaded, a marginal note is placed at the start of their descriptions.

4. PDF output is taken for granted throughout the book, in addition to the classical DVI format. This means that the added possibilities of pdfTeX and `dvipdfm` are explained where they are relevant. A separate Chapter 13 on PostScript and PDF is still necessary, and the `hyperref` package, the best interface for PDF output with all its bells and whistles, is explained in detail. PDF is also included in Chapter 17 on presentation material.

   On the other hand, the other Web output formats, HTML and XML, are only dealt with briefly in Appendix E, since these are large topics treated in other books, most noticeably the *LaTeX Web Companion*.

5. This book is being distributed with a modified version of one of the CDs from the TeX Live set. It contains a full TeX and LaTeX installation for Windows, Macintosh OSX, and Linux, plus many of the myriad extensions that exist.

We once again express our hope that this *Guide* will prove more than useful to all those who wish to find their way through the intricate world of LaTeX. And with the addition of the included TeX Live CD, that world is brought even closer to their doorsteps.

Helmut Kopka and Patrick W. Daly
September 2003

Part I

Basics

This page intentionally left blank

1 Introduction

1.1 Just what is LaTeX?

This book is a manual for LaTeX, a *documentation preparation system* widely used in the fields of mathematics and natural sciences, but which is also spreading to many other disciplines. A brief overview of its features is presented here.

- LaTeX is a comprehensive set of markup commands used with the powerful typesetting program TeX for the preparation of a wide variety of documents, from scientific articles, to reports, to complex books.
- LaTeX, like TeX, is a totally open software system, available free of charge. It is also permissible to modify and redistribute all or any part of it, provided the name is altered, in order to avoid confusion.
- The LaTeX core is maintained by the LaTeX3 Project Group but it also benefits from extensions written by hundreds of user/contributors, with all the advantages and disadvantages of such a democracy.
- A LaTeX document consists of one or more source files containing plain text characters, the actual textual content plus markup commands. These include instructions that can insert graphical material produced by other programs.
- It is processed by the TeX program to produce a binary file in DVI (*device independent*) format containing precise directions for the typesetting of each character. This in turn can be viewed on a monitor, or converted into printer instructions, or some other electronic form such as PostScript, HTML, XML, or PDF.
- A variant of the TeX program called pdfTeX produces PDF output directly from the source file without going through the DVI intermediary. With this, LaTeX can automatically include internal links and bookmarks with little or no extra effort, plus PDF buttons and external links, in addition to graphics in a wide range of common formats.

- TEX activities are coordinated by the TEX Users Group, TUG (`www.tug.org`), as well as by many other national groups (see `www.tug.org/usergroups.html`). Contributions are made by anyone who feels he or she has something vital (or simply interesting) to add.

- TEX and LATEX installations are provided online through the *Comprehensive TEX Archive Network*, CTAN (Section B.4), where all contributed programs and source files may also be found. In addition, there is a set of CDs and a DVD, known as TEX Live, containing the current versions of the installations for various computer types (`www.tug.org/texlive/`). These are distributed annually to members of the TEX groups.

The rest of this book attempts to fill in the gaps in the previous summary. With the help of the included CD from the TEX Live set, which also contains a directory specific to this book (`books→Guide4`), we hope to give the user additional pleasure in learning the joys of LATEX.

1.2 Markup Languages

1.2.1 Typographical markup

In the days before computers, an author would prepare a *manuscript* either by hand or by typewriter, which he or she would submit to a publisher. Once accepted for publication (and after several rounds of corrections and modifications, each requiring a rewrite of the paper manuscript), it would be sent to a copy editor, a human being who would 'decorate' the manuscript with *edits* and *markup*, marginal notes that tell the typesetter (another human being) which fonts, spacing, and other typographical features should be used to convert it to the final printed form of a book or article.

Electronic processing of text today follows a similar procedure, except that the humans have been replaced by computer programs. (So far the author has mainly avoided this fate, but the publishing industry is rapidly working on it.) The markup is normally included directly in the manuscript in such a way that it is converted immediately to its output form and displayed on the computer monitor. This is known as WYSIWYG, or 'what you see is what you get.'

However, what you see is not always what you've got. An alternative that is used more and more by major publishers is *markup languages*, in which the raw text is interspersed with indicators 'for the typesetter.' The result as seen on the monitor is much the same as a typewritten manuscript, except that the markup is no longer abbreviated marginal notes, but cryptic code within the actual text. This *source text*, which can be prepared by a simple, dumb *text editor* program, is converted into typographically set output by a separate program.

For example, to code the line

He took a **bold step** forward.

with HTML, the classical markup language of the World Wide Web, one enters in the source text:

```
He took a <b>bold step</b> forward.
```

In Plain TEX, the same sentence would be coded as:

```
He took a {\bf bold step} forward.
```

The first example is to be processed (displayed) by a Web browser program that decides to set everything between and as boldface. The second example is intended for the TEX program (Section 1.3). The markup in these two examples follows different rules and different syntax, but the functionality is the same.

1.2.2 Logical markup

The above examples illustrate *typographical markup*, in which the inserted commands or tags give direct instructions to alter the appearance of the output, here a change of font. An alternative is to indicate the purpose of the text. For example, HTML recognizes several levels of headings; to place a title into the highest level one enters:

```
<h1>Logical Markup</h1>
```

The equivalent LATEX entry would be:

```
\title{Logical Markup}
```

With this *logical markup*, the author concentrates entirely on the content and leaves the typographical considerations to the experts. One merely marks the structure of the document, without worrying about how the logical elements, such as section titles, are to be rendered typographically. This information is put into HTML style sheets or LATEX classes and packages, which are external to the actual source file or at least restricted to its preamble. It may even be the author who undertakes this task, but such that the typographic design is separated from the actual content.

Today much effort is being put into XML, the Extensible Markup Language, as the ultimate markup system, since it allows the markup, or tags, to be defined as needed, without any indication of how they are to be implemented. That is left to XSL, the Extensible Stylesheet Language. It must be emphasized that neither XML nor XSL are programs at all; they are specifications for how documents and databases may be marked up, and how the markup tags may be translated into real output. Programs still need to be found to do the actual job.

And this is the fundamental idea behind markup languages: that the source text indicates the logical structure of its contents. Such source files, being written in plain ASCII text, are extremely robust, not being married to any particular software package or computer type.

What does all this have to do with LATEX? In the next section we outline the development of TEX and LATEX, and go on to show that LATEX, a product of the mid-1980s, is a programmable markup language that is ideally suited for the modern world of electronic publishing.

1.3 TEX and its offspring

The most powerful formatting program for producing book-quality text of scientific and technical works is that of Donald E. Knuth (Knuth, 1986a, 1986b, 1986c, 1986d, 1986e). The program is called TEX, which is a rendering in capitals of the Greek letters $\tau\epsilon\chi$. For this reason the last letter is pronounced not as an *x*, but as the *ch* in Scottish *loch* or German *ach*, or as the Spanish *j* or Russian *kh*. The name is meant to emphasize that the printing of mathematical texts is an integral part of the program and not a cumbersome add-on. In addition to TEX, the same author has developed a further program called METAFONT for the production of character fonts in electronic form. The standard TEX program package contains 75 fonts in various design sizes, each of which is also available in up to eight magnification steps. All these fonts were produced with the program METAFONT. With additional applications, additional character fonts have been created, such as for Cyrillic, Chinese, and Japanese, with which texts in these alphabets can be printed in book quality.

The TEX program is free, and the source code is readily available. Anybody may take it and modify it as they like, provided they call the result something other than TEX. This indeed has occurred, and several TEX variants do exist, including pdfTEX which we deal with later in this chapter. Only Knuth is allowed to alter TEX itself, which he does only to correct any obvious bugs. Otherwise he considers TEX to be completed; the current version number is 3.141592, and with his death the code will be frozen for all time, and the version number will become exactly π.

1.3.1 The TEX program

The basic TEX program only understands a set of very primitive commands that are adequate for basic typesetting operations and programming functions. However, it does allow more complex, higher-level commands to be defined in terms of the primitive ones. In this way, a more user-friendly environment can be constructed out of the low-level building blocks.

During a processing run, the program first reads in what is known as a *format file*, which contains the definitions of the higher-level commands in terms of the primitive ones as well as the hyphenation patterns for word division. Only then does it read in the author's *source file* containing the actual text to be processed, including formatting commands that are predefined in the format file.

Creating new formats should be left to very knowledgeable programmers. The definitions are written to a source file which is then processed with a special version of the TEX program called initex. It stores the new format file in a compact manner so that it can be read in quickly by the regular TEX program.

Although the normal user will almost never write such a format, he or she may be presented with a new format source file that will need to be installed with initex. For example, this is just what must be done to upgrade LATEX periodically. How to do this is described in Appendix B.

1.3.2 Plain TEX

Knuth has provided a basic format named *Plain TEX* to interact with TEX at its simplest level. This is such a fundamental part of TEX processing that one tends to forget the distinction between the actual processing program TEX and this particular format. Most people who claim to 'work only with TEX' really mean that they only work with Plain TEX.

Plain TEX is also the basis of every other format, something that only reinforces the impression that TEX and Plain TEX are one and the same.

1.3.3 LATEX

The emphasis of Plain TEX is still very much at the typesetter's level rather than the author's. Furthermore, the exploitation of all its potential demands considerable experience with programming techniques. Its application thus remains the exclusive domain of typographic and programming professionals.

For this reason, the American computer scientist Leslie Lamport has developed the LATEX format (Lamport, 1985), which provides a set of higher-level commands for the production of complex documents. With it, even the user with no knowledge of typesetting or programming is in a position to take extensive advantage of the possibilities offered by TEX and to be able to produce a variety of text outputs in book quality within a few days, if not hours. This is especially true for the production of complex tables and mathematical formulas.

As pointed out in Section 1.2.2, LATEX is very much more a *logical* markup language than the original Plain TEX on which it is based. It contains provisions for automatic running heads, sectioning, tables of contents, cross-referencing, equation numbering, citations, and floating tables and figures, without the author having to know just how these are to be formatted. The layout information is stored in additional *class files* that are referred to but not included in the input text. The predefined layouts may be accepted as they are or replaced by others with minimal changes to the source file.

Since its introduction in the mid-1980s, LATEX has been periodically updated and revised, like all software products. For many years the version number was fixed at 2.09 and the revisions were only identified by their dates. The last major update occurred on December 1, 1991, with some minor corrections up to March 25, 1992, at which point LATEX 2.09 became frozen.

1.3.4 LATEX 2ε

The enormous popularity of LATEX and its expansion into fields for which it was not originally intended, together with improvements in computer technology, especially with regard to cheap but powerful laser printers, had created a diversity of formats bearing the LATEX label. In an effort to reestablish a genuine, improved standard, the LATEX3 Project was set up in 1989 by Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. Their goal was to construct an optimized and efficient set of

basic commands complemented by various *packages* to add specific functionality as needed.

As the name of the project implies, its aim is to achieve a version 3 for LaTeX. However, since that is the long-term goal, a first step towards it was the release of LaTeX 2ε in mid-1994 together with the publication of the second edition of Lamport's basic manual (Lamport, 1994) and of an additional book, *The LaTeX Companion*, (Goossens et al., 1994) describing many of the extension packages available and LaTeX programming in the new system. Since then, two further books have appeared, Goossens et al. (1997), covering the inclusion of graphics and color, and Goossens and Rahtz (1999), explaining how LaTeX may be used with the World Wide Web. A completely reworked and extended 2nd edition of *The LaTeX Companion* by Mittelbach et al. (2004) has now become available.

Initially, updates to LaTeX 2ε were issued twice a year, but it has now become so stable that since 2000 new releases have been much less frequent. The latest, dated December 2003, appeared in March 2004, after a break of over two years.

LaTeX 2ε is now the standard version, and LaTeX 2.09 is considered obsolete, although source files intended for the older version may still be processed with the newer one. In this book, unless otherwise indicated, 'LaTeX' will always mean LaTeX 2ε.

1.3.5 TeX fonts

TeX initially made use of its own set of fonts, called Computer Modern, generated by Knuth's METAFONT program. The reason for doing this was that printers at that time (and even today) may contain their own preloaded fonts, but they are often slightly different from printer to printer. Furthermore, they lacked the mathematical character sets that are essential to TeX's main hallmark, mathematical typesetting. So Knuth created pixel fonts that could be sent to every printer, thus ensuring the same results everywhere.

Today the situation with fonts has changed dramatically. Outline fonts (also known as type 1 fonts) are more compact and versatile than the pixel fonts (type 3). They also have a far superior appearance and are drawn much faster in PDF files. The original Computer Modern fonts have been converted to outline fonts, but there is no reason to stick with them, except possibly for the mathematical symbols. For example, the 35 standard PostScript fonts are now a basic part of any LaTeX installation (Section 13.1.2). It is LaTeX 2ε with its New Font Selection Scheme that freed TeX from its rigid marriage to Computer Modern.

1.3.6 The LaTeX bazaar: user contributions

Like the TeX program on which it relies, LaTeX is freeware. There may be a prejudice that what is free is not worth anything, but there are several other examples in the computer world (for example, GNU, Linux) to contradict this statement. And since the LaTeX macros are provided in files containing plain text, there is no problem to exchange, modify, and supplement them. In other words, the user can participate in extending the basic LaTeX system.

Taking advantage of a mechanism in LATEX 2.09 that allowed options to the default layouts to be contained in so-called *style option files*, many users began writing their own 'options' to provide additional features to the basic LATEX. They then made these available to other users via the Internet. Many were intended for very specific problems, but many more proved to be of such general usefulness that they have become part of the standard LATEX installation. In this way, the users themselves have built up a system that meets their needs.

With LATEX 2ε, these user contributions acquired official status: They became known as *packages*, they could be entered directly into the document and not by the back door, guidelines were issued for writing them, and additional commands were introduced to assist package programming. Package files bear the extension `.sty` from LATEX 2.09 days, so that the older style option files may still function as packages today.

Contributions may also be made as *class* files, something that defines the overall structure of the document. These correspond to the old LATEX 2.09 main style files, which may not be reused as class files anymore. Most contributed class files start by loading one of the standard ones and then making modifications to it.

Those packages and classes that have established themselves as indispensable for sophisticated LATEX processing are described in this book in those sections where they are most relevant. This does not imply that the others are less worthwhile, but simply that this book does have to make a selection. Many other packages are described fully in *The LATEX Companion* (Mittelbach et al., 2004).

1.3.7 The LATEX Project Public License

The LATEX core and most of the contributions are distributed under the conditions of the LATEX Project Public License (LPPL). The contents of the TEX Live distribution are subject to the terms of this license, a rigid requirement for their inclusion. The main purpose of LPPL is to ensure the free distribution of the software and support files, while guaranteeing the integrity of naming schemes so that all users employing version *n* of package *X* will obtain identical results. Thus it is allowed to modify packages and programs provided they are given a new name.

The current text of LPPL can be found at

```
www.latex-project.org/lppl.txt
```

1.3.8 LATEX and electronic publishing

The most significant development in computer usage in the last decade is the rise of the World Wide Web (or the hijacking of the Internet by the advertising world). LATEX makes its own contribution here with

- programs to convert LATEX files to HTML (Appendix E);
- means of creating PDF output, with hypertext features such as links, bookmarks, and active buttons (Chapter 13);

- interfacing to XML both by acting as an engine to render XML documents and with programs to convert LaTeX to XML and vice versa (Appendix E).

All these forms of electronic publishing are alternatives to traditional paper output. We do not expect paper to disappear entirely so quickly, but it is rapidly being replaced by electronic forms, which can always reproduce the paper whenever needed.

1.4 How to use this book

This book is meant to be a mixture of textbook and reference manual. It explains all the essential elements of the current standard LaTeX 2_ε, but compared to Lamport (1985, 1994), it goes into more detail, offers more examples and exercises, and describes many 'tricks' based on the authors' experiences. It explains not only the core LaTeX installation, but also many of the contributed packages that have become essential to modern LaTeX processing, and thus quasi-standard. We necessarily have to be selective, for we cannot go to the same extent as *The LaTeX Companion* (Mittelbach et al., 2004), *The LaTeX Graphics Companion* (Goossens et al., 1997), and *The LaTeX Web Companion* (Goossens and Rahtz, 1999), which are still valid *companions* to this book.

The first part of the book is entitled *The Basics*, and covers the more fundamental aspects of LaTeX: inputting text and symbols, document organization, lists and tables, entering mathematics, and customizations by the user. The second part is called *Beyond the Basics*, meaning it presents concepts that may be more advanced but are still essential to producing complex, sophisticated documents. The distinction is rather arbitrary. Finally, the appendices contain topics that are not directly part of LaTeX itself, but useful for understanding its applications: installation, error messages, creating packages, World Wide Web, and fonts. Appendix G is an alphabetized summary of most of the commands and their use, cross-referenced to their locations in the main text.

1.4.1 Some conventions

In the description of command syntax, `typewriter` type is used to indicate those parts that must be entered exactly as given, while *italic* is reserved for those parts that are variable or for the text itself. For example, the command to produce tables is presented as follows:

`\begin{tabular}{`*col_form*`}` *lines* `\end{tabular}`

The parts in typewriter type are obligatory, while *col_form* stands for the definition of the column format that must be inserted here. The allowed values and their combinations are given in the detailed descriptions of the commands. In the above example, *lines* stands for the line entries in the table and are thus part of the text itself.

Package: `sample`

Sections describing a package, an extension to basic LaTeX, have the name of that package printed as a marginal note, as demonstrated here for this paragraph. In this way, you are reminded that you must include it with `\usepackage` (Section 3.1.2) in order to obtain these additional features.

Sections of text that are printed in a smaller typeface together with the boxed exclamation mark at the left are meant as an extension to the basic description. They may be skipped over on a first reading. This information presents deeper insight into the workings of LaTeX than is necessary for everyday usage, but which is invaluable for creating more refined control over the output.

1.5 Basics of a LaTeX file

1.5.1 Text and commands

The *source file* for LaTeX processing, or simply the *LaTeX file*, contains the *source text* that is to be processed to produce the printed output. Splitting the text into lines of equal width, formatting it into *paragraphs*, and breaking it into *pages* with page numbers and running heads are all functions of the processing program and not of the input text itself.

For example, words in the source text are strings of letters terminated by some non-letter, such as *punctuation*, *blanks*, or *end-of-lines* (*hard* end-of-lines, ones that are really there, not the *soft* ones that move with the window width); whereas punctuation marks will be transferred to the output, blanks and end-of-lines merely indicate a gap between words. Multiple blanks in the input, or blanks at the beginning of a line, have no effect on the interword spacing in the output.

Similarly, a new paragraph is indicated in the input text by an empty line; multiple empty lines have the same effect as a single one. In the output, the paragraph may be formatted either by indentation of the first line or by extra interline spacing, but this is not affected in any way by the number of blank lines or extra spaces in the input.

The source file contains more than just text, however; it is also interspersed with markup commands that control the formatting or indicate the structure. It is therefore necessary for the author to be able to recognize what is text and what is a command. Commands consist either of certain single characters that cannot be used as text characters or of words preceded immediately by a special character, the backslash (\).

The syntax of source text is explained in detail in Chapter 2.

1.5.2 Contents of a LaTeX source file

Every LaTeX file contains a *preamble* and a *body*.

The preamble is a collection of commands that specify the global processing parameters for the following text, such as the paper format, the height and width of the text, and the form of the output page with its pagination and automatic

page heads and footlines. At a minimum, the preamble must contain the command `\documentclass` to specify the document's overall processing type. This is the first command in the preamble.

If there are no other commands in the preamble, LaTeX selects standard values for the line width, margins, paragraph spacing, page height and width, and much more. By default, these specifications are tailored to the American norms. For European requirements, built-in options exist to alter the text height and width to the A4 standard. Furthermore, there are language-specific packages to translate certain headings, such as 'Chapter' and 'Abstract'.

The preamble ends with `\begin{document}`. Everything that follows this command is interpreted as *body*. It consists of the actual text mixed with markup commands. In contrast to those in the preamble, these commands have only a local effect, meaning they apply only to a part of the text, such as *indentation*, *equations*, temporary change of *font*, and so on. The body ends with the command `\end{document}`. This is normally the end of the file as well.

The general syntax of a LaTeX file is as follows:

```
\documentclass[options]{class}
    Further global commands and specifications
\begin{document}
    Text mixed with additional commands of local effect
\end{document}
```

The possible *options* and *classes* that may appear in the `\documentclass` command are presented in Section 3.1.1.

A minimal LaTeX file named `hi.tex` contains just the following lines:

```
\documentclass{article}
\begin{document}
   Hi!
\end{document}
```

A more complex sample file is shown in Section 1.7.

1.5.3 Extending LaTeX with packages

Packages are a very important feature of LaTeX. These are extensions to the basic LaTeX commands that are written to files with names that end in `.sty` and are loaded with the command `\usepackage` in the preamble. Packages can be classified by their origin.

- **Core** packages are an integral part of the LaTeX basic installation and are therefore fully standard.
- **Tools** packages are a set written by members of the LaTeX3 Team and should always be in the installation.

- **Graphics** packages are a standardized set for including pictures generated by other programs and for handling color; they are on the same level as the tools packages.
- $\mathcal{AMS}$-LaTeX packages, published by the American Mathematical Society, should be in any installation.
- **Contributed** packages have been submitted by actual users; certain of these have established themselves as 'essential' to standard LaTeX usage, but all are useful.

Only a limited number of these packages are described in this book, those that we consider indispensable. However, there is nothing to prevent the user from obtaining and incorporating any others that should prove beneficial for his or her purposes.

There are more than 1000 contributed packages on the TeX Live distribution. How can one begin to get an overview of what they offer? Graham Williams has compiled a list of brief descriptions that can be found online and on TeX Live at

`texmf→doc→html→catalogue→catalogue.html`

(Here and elsewhere in this book, we give paths on the CD with → as the directory separator, representing the \ for Windows, / for Unix, and : for Macintosh.)

How to load packages into the LaTeX source file is explained in Section 3.1.2.

Documentation of contributed packages is somewhat haphazard depending on how much the author has put into it. The preferred method for distributing packages is to integrate the documentation with the code into a single file with extension `.dtx`. A special program DocStrip is used to extract the actual package file or files, while LaTeXing the original `.dtx` file produces the instruction manual. Most ready-to-run installations will already have done all this for the user, with the resulting manuals stored as DVI or PDF files somewhere in `texmf→doc→latex→...`. However, you might have to generate the documentation output yourself by processing the `.dtx` file, which should be found in `texmf→source→latex→...` (Section B.3 explains the organization of the TeX directory system.)

Some package authors write their manuals as an extra `.tex` file, the output of which may or may not be prestored in DVI or PDF form. Others provide HTML files. And still others simply add the instructions as comments in the package file itself. (This illustrates some of the joys of an open system.)

1.6 TEX processing procedure

Since LaTeX is a set of definitions for the TeX program, LaTeX processing itself is in fact TeX processing with the LaTeX format. What TeX does with this is the same as for any other of the many formats available (of which LaTeX is perhaps the most popular). All the typesetting work is done by TeX, while LaTeX handles the conversion from the logical markup to the typesetting commands. It also enables cross-referencing, running headlines, table of contents, literature citations and bibliography, indexing,

and more. However, the processing of the source file to final output is TEX's task regardless of the format being used.

1.6.1 In the good old days

TEX arose more than 20 years ago before there were such things as PCs and graphical displays, and before computers were 'infected' with windows or mice. TEX and its support programs were invoked from a command line, not with a mouse click. This may sound very old fashioned, but it did guarantee portability to all computer types.

The processing steps that were taken in those days still exist with today's graphical interfaces, but are now executed more conveniently. One can still open a 'command prompt window' and run them from the command line.

The first step is, of course, to use a *text editor* program to write the source file containing the actual text and markup. The rules for entering this source text are explained in Chapter 2. It goes into a text file, or what is often called an ASCII file containing only standard punctuation marks, numbers, unaccented letters, and upper- and lowercase. In other words, the text is that which can be produced from a standard English typewriter.

The name of the source file normally has the extension `.tex`; it is then processed by TEX to produce a new file with the same base name and the extension `.dvi`, for *device independent* file. This is a binary file (all codes possible, not a text file) containing precise instructions for the selection and placement of every symbol, a coded description of the final printed page. The command to invoke TEX with the source file `hi.tex` is

```
tex &latex hi
```

meaning run the TEX program with the format `latex`. Usually the installation has defined a shortcut named `latex` to do this, so

```
latex hi
```

should be sufficient. It is only necessary to specify the extension of the source file name if it is something other than `.tex`.

During the processing, TEX writes information, warnings, error messages to the computer monitor, and to a *transcript* file with the extension `.log`. It is well worth inspecting this file when unexpected results appear.

The final step is to produce the printed pages from the DVI file. This requires another program, a *driver*, to generate the instructions specific to the given printer. For example, to produce a PostScript file, run

```
dvips hi
```

to obtain `hi.ps` from `hi.dvi`. And then send `hi.ps` to the PostScript printer with the regular command for that computer system.

Previewing the DVI file on a computer monitor before printing was a later development, requiring high-quality graphics displays. These programs are essentially special drivers that send the output directly to the monitor rather than to a printer or printer file. One very popular previewer is called with

```
xdvi hi
```

to view `hi.dvi` before committing it to paper.

1.6.2 And today

The various steps for LATEX processing described above are still necessary today, and one can open up a command prompt window and carry them out just as before. However, intelligent editors with LATEX-savvy now exist that not only assist writing the source text, but also will call the various programs—TEX, previewer, printer driver, BIBTEX, MakeIndex (these are explained later)—with a mouse click.

One such editor for Windows is called *WinShell*, written by Ingo H. de Boer (`www.winshell.de`). Although free of charge, its author appreciates donations to offset his expenses.

Another such editor and LATEX interface is *WinEdt* by Aleksander Simonic (`www.winedt.com`). A sample window with the text from the demonstration file in Section 1.7 is shown in Figure 1.1. This program is available for a 30-day trial period, after which one must pay a nominal fee to obtain a license. It is the editor that we ourselves use and we can highly recommend it.

An alternative is LyX, a free open source software for document processing that is almost WYSIWYG, acting as a front-end to LATEX, where the user need not know anything about LATEX. See its home page at `www.lyx.org`.

It must be stressed that all the above are *interfaces* to an existing LATEX installation. On the other hand, there are also commercial packages that include both the TEX/LATEX installation and a graphics interface. These are listed in Section B.1.1.

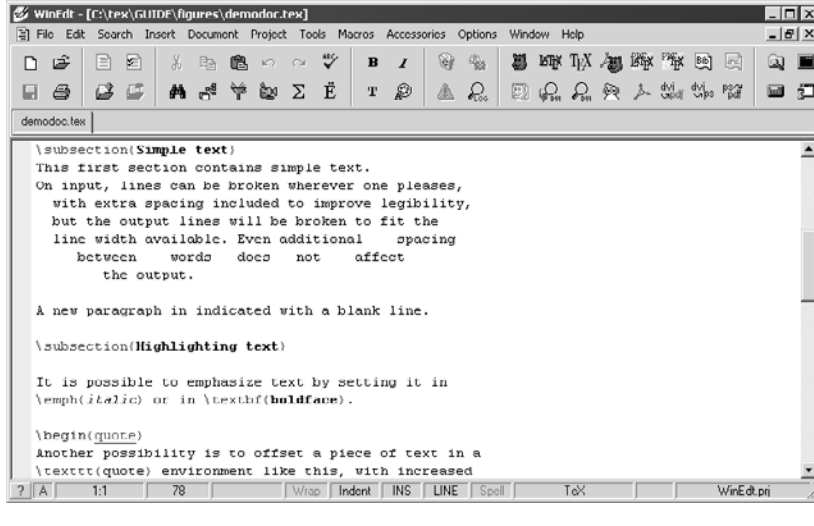


Figure 1.1: Sample display with the WinEdt editor for interfacing to LATEX

1.6.3 Alternative to TEX: pdfTEX

As we mentioned earlier, it is permissible to use the TEX source code to generate something else, as long as it bears another name. One such modification is called pdfTEX, created by Hàn Thế Thành. This program does everything TEX does, but it optionally writes its output directly to a PDF file, bypassing the DVI output of regular TEX. It therefore combines the TEX program with a DVI-to-PDF driver program. Normally this option is also the default.

There are many advantages to producing PDF output directly this way, apart from saving a step. The PDF file is generated in exactly the same way as the DVI file with TEX and can be viewed immediately with the Acrobat Reader or other PDF viewer. The results can be sent directly to a printer without going through the DVI-to-Printer program. It is also much easier to include the hypertext features of a true active PDF file, as we explain in Section 13.2.4.

Adding the LATEX macros to pdfTEX produces something one could call pdfLATEX. This distinction is only meaningful for invoking the program-plus-format to process the LATEX source file. Except for some things that we note in Section 13.2.3, LATEX commands are identical whether used with TEX or with pdfTEX. This makes the conversion extremely easy.

The rest of this book deals essentially with LATEX itself, regardless of what the end product is to be: paper, HTML, XML, or PDF.

1.7 Sample LATEX file

Here we present a more complicated sample LATEX file, which can be found on the TEX Live CD at `books→Guide4→demodoc.tex`. The annotations in the following source text give the section numbers where the various features are explained in detail. The resulting output is shown in Figure 1.2.

```
\documentclass[a4paper,12pt]{article}
\usepackage{palatino}
\usepackage{graphicx}
\usepackage{amsmath}
\usepackage[margin=1.5cm,
  vmargin={0pt,1cm},
  includefoot]{geometry}
\title{Demonstration Document}
\author{M. Y. Self\\
        At Home}
\date{February 1, 2004}

% End of preamble,
% the body now follows
```

Mandatory, document type and options (Sec. 3.1)

Loading extra features (Sec. 3.1.2):–
Use Palatino fonts (Sec. 13.1.2)
Enable graphics importation (Chap. 8)
Add extra math functions (Chap. 15)
Add page layout specs. (Sec. 3.2.6)

Enter texts for title block (Sec. 3.3.1). The author text contains name with affiliation on 2nd line.

Comment lines beginning with % are ignored in processing (Sec. 4.7).

Demonstration Document

M. Y. Self
At Home

February 1, 2004

1 Introduction

1.1 Simple text

This first section contains simple text. On input, lines can be broken wherever one pleases, with extra spacing included to improve legibility, but the output lines will be broken to fit the line width available. Even additional spacing between words does not affect the output.

A new paragraph in indicated with a blank line.

1.2 Highlighting text

It is possible to emphasize text by setting it in *italic* or in **boldface**.

> Another possibility is to offset a piece of text in a `quote` environment like this, with increased margins on either side.

2 Sophistications

2.1 Mathematics

There are special rules for entering math, and many commands that only exist in math mode. An in-line math formula like $x^2 + \beta$ has automatic spacing between variables and operators, while displayed equations like

$$\vec{A} \times (\vec{B} \times \vec{C}) = (\vec{A} \cdot \vec{C})\vec{B} - (\vec{A} \cdot \vec{B})\vec{C} \tag{1}$$

$$\vec{A} \cdot (\vec{B} \times \vec{C}) = \begin{vmatrix} A_x & A_y & A_y \\ B_x & B_y & B_y \\ C_x & C_y & C_y \end{vmatrix} \tag{2}$$

are given automatic numbers.

2.2 Lists

1. Various lists are possible, to enumerate,
2. to itemize with bullets,
3. or to make descriptive lists.

2.3 Adding graphics

Pictures and graphics  produced by other programs are easily inserted.

Figure 1.2: Output of the demonstration document

```
\begin{document}
\maketitle
\section{Introduction}
\subsection{Simple text}
This first section contains simple text.
On input, lines can be broken wherever one pleases,
  with extra spacing included to improve legibility,
  but the output lines will be broken to fit the
  line width available. Even additional    spacing
     between    words   does   not    affect
        the output.

A new paragraph in indicated with a blank line.

\subsection{Highlighting text}

It is possible to emphasize text by setting it in
\emph{italic} or in \textbf{boldface}.

\begin{quote}
Another possibility is to offset a piece of text in a
\texttt{quote} environment like this, with increased
margins on either side.
\end{quote}

\section{Sophistications}

\subsection{Mathematics}

There are special rules for entering math, and many
commands that only exist in math mode. An
in-line math formula like $x^2+\beta$
has automatic spacing between variables and
operators, while displayed equations like
\begin{equation}
  \vec{A} \times (\vec{B}\times\vec{C}) =
  (\vec{A}\cdot\vec{C})\vec{B} -
  (\vec{A}\cdot\vec{B})\vec{C}
\end{equation}
\begin{equation}
  \vec{A}\cdot(\vec{B}\times\vec{C}) =
  \begin{vmatrix}
    A_x & A_y & A_y \\
    B_x & B_y & B_y \\
    C_x & C_y & C_y
   \end{vmatrix}
```

Start of document body

Generates title block (Sec. 3.3.1)

Make section headings (Sec. 3.3.3)

Two paragraphs

Font styles (Sec. 4.1)

Two-sided indenting (Sec. 4.2.3)

Math mode, Chapters 7 and 15

A $ sign toggles math

Start displayed, numbered equation (Sec. 7.1)

Start 2nd equation

Matrix produced with $\mathcal{A}\mathcal{M}\mathcal{S}$ extension (Sec. 15.2.4)

```
\end{equation}
are given automatic numbers.

\subsection{Lists}

\begin{enumerate}
  \item Various lists are possible, to enumerate,
  \item to itemize with bullets,
  \item or to make descriptive lists.
\end{enumerate}

\subsection{Adding graphics}

Pictures and graphics
\includegraphics[width=1cm]{demo}
produced by other programs are easily inserted.
\end{document}
```

Start numbered list (Sec. 4.3)

Items in list

Insert graphics file (Sec. 8.1.3)

End of entire document

Write the above source text to a file named `demodoc.tex`, and process it with

```
latex demodoc
dvips demodoc
```

for DVI and PS output, or with

```
pdflatex demodoc
```

for PDF output. The results should look like that in Figure 1.2 (except for possible variations due to different versions of the `geometry` package).

Note: In order to fit the entire output on to a single page, it is necessary to decrease the margins considerably over the standard values. The package `geometry` has been loaded with selected values to do this. You may try removing these lines to see what the output would be like with the standard page layout. And if you are using American letter paper, replace the option `a4paper` with `letterpaper` to center the text properly.

This page intentionally left blank

2 Text, Symbols, and Commands

The text that is to be the input to a LaTeX processing run is written to a *source file* with a name ending in `.tex`, the file name extension. This file is prepared with a *text editor*, either one that handles straightforward plain text or one that is configured to assist the writing and processing of LaTeX files. In either case, the contents of this file are plain ASCII characters only, with no special symbols, no accented letters, preferably displayed in a fixed-width typewriter font, with no frills like bold or italics, all in one size. All these aspects of true typesetting are produced afterwards by the TeX processing program with the help of *markup* commands inserted visibly into the actual text. It is therefore vital to know how commands are distinguished from text that is to be printed and, of course, how they function.

(However, for languages other than English, native keyboard input may indeed be used, as shown in Section 2.5.9.)

2.1 Command names and arguments

A *command* is an instruction to LaTeX to do something special, such as print some symbol or text not available to the restricted character set used in the input file, or change the current typeface or other formatting properties. There are three types of command names:

- the single characters `# $ & ~ _ ^ % { }` all have special meanings that are explained later in this chapter;
- the backslash character \ plus a single non-letter character; for example, `\$` to print the $ sign; all the special characters listed above have a corresponding two-character command to print them literally;
- the backslash character \ plus a sequence of letters, ending with the first non-letter; for example, `\large` to switch to a larger typeface. Command names are case sensitive, so `\large`, `\Large` and `\LARGE` are distinct commands.

Many commands operate on some short piece of text, which then appears as an *argument* in curly braces following the command name. For example, `\emph{stress}`

is given to print the word `stress` in an emphasized typeface (here italic) as *stress*. Such arguments are said to be *mandatory* because they must always be given.

Some commands take *optional* arguments, which are normally employed to modify the effects of the command somehow. The optional arguments appear in square brackets.

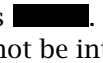
In this book we present the general syntax of commands as

`\`*name*`[`*optional*`]{`*mandatory*`}`

where typewriter characters must be typed exactly as illustrated and italic text indicates something must be replaced. Optional arguments are put into square brackets `[ ]` and the mandatory ones into curly braces `{ }`. A command may have several optional arguments, each one in its set of brackets in the specified sequence. If none of the optional arguments is used, the square brackets may be omitted. Any number of blanks, or even a single new line, may appear between the command name and the arguments to improve legibility.

Some commands have several mandatory arguments. Each one must be put into a `{ }` pair and their sequence must be maintained as given in the command description. For example,

`\rule[`*lift*`]{`*width*`}{`*height*`}`

produces a black rectangle of size *width* and *height*, raised by an amount *lift* above the current baseline. A rectangle of width 10 mm and height 3 mm is made with `\rule{10mm}{3mm}`. Since the optional argument *lift* is omitted, the rectangle is set on the baseline with no lifting, as  . The arguments must appear in the order specified by the syntax and may not be interchanged.

Some commands have a so-called *-form in addition to their normal appearance. A * is added to their name to modify their functionality somehow. For example, the `\section` command has a *-form `\section*` which, unlike the regular form, does not print an automatic section number. For each such command, the difference between the normal and *-form will be explained in the description of the individual commands.

Command names consist only of letters, with the first non-letter indicating the end of the name. If there are optional or mandatory arguments following the command name, then it ends before the `[` or `{` bracket, since these characters are not letters. Many commands, however, possess no arguments and are composed of only a name, such as the command `\LaTeX` to produce the LaTeX logo. If such a command is followed by a punctuation mark, such as comma or period, it is obvious where the command name ends. *However, any blanks following the command name will also terminate it, but are then swallowed up:* `The \LaTeX logo` results in 'The LaTeXlogo', that is, the blank was seen only as the end of the command and not as spacing between two words. This is a result of the special rules for blanks, described in Section 2.5.1.

To insert a space after a command that consists only of a name, either an empty structure `{}` or a space command (\ and blank) must be placed after the command. The proper way to produce 'The LaTeX logo' is to type either `The \LaTeX{} logo` or

`The \LaTeX\ logo`. Alternatively, the command itself may be put into curly braces, as `The {\TeX} logo`, which also yields the right output with the inserted blank: 'The TEX logo'. Incidentally, the LATEX 2ε logo is produced with `\LaTeXe`. Can you see why this logo command cannot be named `\LaTeX2e`?

2.2 Environments

An *environment* is initiated with the command `\begin{`*name*`}` and is terminated by `\end{`*name*`}`.

An environment affects the text within it by treating it differently according to the environment parameters. It is possible to alter (temporarily) certain processing features, such as indentation, line width, typeface, and much more. The changes apply only within the environment. For example, with the `quote` environment

```
previous text
\begin{quote}
text1 \small text2 \bfseries text3
\end{quote}
following text
```

the left and right margins are increased relative to those of the previous and following texts. In the example, this applies to the three texts *text1*, *text2*, and *text3*. After *text1* comes the command `\small`, which has the effect of setting the next text in a smaller typeface. After *text2*, there is an additional command `\bfseries` to switch to boldface type. Both commands only remain in effect up to the `\end{quote}`.

> The three texts within the `quote` environment are indented on both sides relative to the previous and following texts. The *text1* appears in the normal typeface, the same one as outside the environment. The *text2* and *text3* appear in a smaller typeface, and *text3*, **furthermore, appears in boldface.**

After the end of the `quote` environment, the subsequent text appears in the same typeface that was in effect beforehand.

Note that if the names of the environment in the `\begin{..}` `\end{..}` pair do not match, an error message will be issued on processing.

Most declaration command names (see next section) may also be used as environment names. In this case the command name is used *without* the preceding `\` character. For example, the command `\em` switches to an emphatic typeface, usually *italic*, and the corresponding environment `\begin{em}` will set all the text in *italic* until `\end{em}` is reached.

A nameless environment can be simulated by a `{...}` pair. The effect of any command within it ends with the closing curly brace.

You can even create your own environments, as described in Section 10.4.

2.3 Declarations

A *declaration* is a command that changes the values or meanings of certain parameters or commands without printing any text. The effect of the declaration begins immediately and ends when another declaration of the same type is encountered. However, if the declaration occurs within an environment or a `{...}` pair, its scope extends only to the corresponding `\end` command or to the closing brace `}`. The commands `\bfseries` and `\small` mentioned in the previous section are examples of such nonprinting declarations that alter the current typeface.

Some declarations have associated arguments, such as the command `\setlength`, which assigns a value to a *length parameter* (see Sections 2.4 and 10.2).

Examples:

`{\bfseries This text appears in boldface}` The `\bfseries` declaration changes the typeface: **This text appears in boldface**. The effect of this declaration ends with the closing brace `}`.

`\setlength{\parindent}{0.5cm}` The paragraph indentation is set to 0.5 cm. The effect of this declaration ends with the next encounter of the command `\setlength{\parindent}` or at the latest with the `\end` command that terminates the current environment.

`\pagenumbering{roman}` The page numbering is to be printed in Roman numerals.

Some declarations, such as the last example, are global; that is, their effects are not limited to the current environment. The following declarations are of this nature, the meanings of which are given later:

```
\newcounter          \pagenumbering      \newlength
\setcounter          \thispagestyle      \newsavebox
\addtocounter
```

Declarations made with these commands are effective right away and remain so until they are overridden by a new declaration of the same type. In the last example above, page numbering will be done in Roman numerals until countermanded by a new `\pagenumbering{arabic}` command.

2.4 Lengths

2.4.1 Fixed lengths

Lengths consist of a decimal number with a possible sign in front (+ or –) followed by a mandatory dimensional unit. Permissible units and their abbreviated names are:

`cm`	centimeter
`mm`	millimeter
`in`	inch (1 in = 2.54 cm)
`pt`	point (1 in = 72.27 pt)
`bp`	big point (1 in = 72 bp)
`pc`	pica (1 pc = 12 pt)
`dd`	didôt point (1157 dd = 1238 pt)
`cc`	cicero (1 cc = 12 dd)
`em`	a font-specific size, the width of the capital M
`ex`	another font-related size, the height of the letter x

Decimal numbers in TEX and LATEX may be written in either the English or European manner, with a *period* or a *comma:* both `12.5cm` and `12,5cm` are permitted.

Note that `0` is not a legitimate length since the unit specification is missing. To give a zero length it is necessary to add some unit, such as `0pt` or `0cm`.

Values are assigned to a length parameter by means of the LATEX command `\setlength`, which is described in Section 10.2 along with other commands for dealing with lengths. Its syntax is:

`\setlength{\`*length_name*`}{`*length_spec*`}`

For example, the width of a line of text is specified by the parameter `\textwidth`, which is normally set to a default value depending on the class, paper type, and font size. To change the line width to be 12.5 cm, one would give:

```
\setlength{\textwidth}{12.5cm}
```

2.4.2 Rubber lengths

Some parameters expect a *rubber* length. These are lengths that can be stretched or shrunk by a certain amount. The syntax for a rubber length is:

nominal_value `plus` *stretch_value* `minus` *shrink_value*

where the *nominal_value, stretch_value*, and *shrink_value* are each a length. For example,

```
\setlength{\parskip}{1ex plus0.5ex minus0.2ex}
```

means that the extra line spacing between paragraphs, called `\parskip`, is to be the height of the *x* in the current font, but it may be increased to 1.5 or reduced to 0.8 times that size.

One special *rubber* length is `\fill`. This has the natural length of *zero* but can be stretched to any size.

2.5 Special characters

2.5.1 Spaces

The *space* or *blank* character has some properties that are different from those of normal characters, some of which have already been mentioned in Section 2.1. During processing, blanks in the input text are replaced by rubber lengths (Section 2.4.2) to allow the line to fill up to the full line width. As a result, some peculiar effects can occur if one is not aware of the following rules:

- One blank is the same as a thousand, only the first one counts.
- Blanks at the beginning of an input line are ignored.
- Blanks terminating a command name are removed.
- The end of a line is treated as a blank.

Some of the consequences of these rules are that there may be as many blanks as desired between words or at the beginning of a line (to make the input text more legible) and that a word may come right at the end of a line without the spacing between it and the next word disappearing. To force a space to appear where it would otherwise be ignored, one must give the command \␣ (a \ followed by a space character, made visible here by the symbol ␣).

To ensure that certain words remain together on the same line, a *protected space* is inserted between them with the ~ character (Section 2.7.1, page 32). Multiple protected spaces are all printed, in contrast to normal spaces.

Sometimes it is necessary to suppress the space that appears because of the new line. In this case, the last character in the line must be the *comment* character % (Section 4.7).

Paragraphs are separated in the source text by blank lines. As for blank characters, one blank line is the same as a thousand. Instead of a blank line, the command `\par` may also be used to indicate the end of a paragraph.

2.5.2 Quotation marks

The *quotation marks* found on the typewriter, ", are not used in book printing. Instead, different characters are used at the beginning and end, such as ‘single quotes’ and “double quotes”. Single quotes are produced with ` and ', while double quotes are made by typing the respective characters twice: `` for “ and '' for ”. Furthermore, the typewriter character " will also generate the double closing quote ”. However, it should be avoided since it can lead to confusion.

2.5.3 Hyphens and dashes

In book printing, the character that appears on the typewriter as - comes in various lengths: -, –, —. The smallest of these, the *hyphen*, is used for compound words such

as *father-in-law* and for word division at the end of a line. The middle-sized one, the *en dash*, is used in ranges of numbers, for example, pages 33–36. And the largest one, the *em dash*, is used as punctuation—what is normally called the *dash*. These are generated by typing the hyphen character one, two, or three times, so that - yields -, while -- makes –, and --- produces —. A fourth type of dash is the minus sign −, which is entered in math mode as `$-$` (Chapter 7).

2.5.4 Printing command characters

As mentioned in Section 2.1, the characters `# $ & _ % { }` are interpreted as commands. To print them as text, one must give a command consisting of \ plus that character.

`#` = `\#` `$` = `\$` `&` = `\&` `_` = `\_` `%` = `\%` `{` = `\{` `}` = `\}`

2.5.5 The special characters §, †, ‡, ¶, ©, and £

These special characters do not exist on the computer keyboard. They can, however, be generated by special commands as follows:

§ = `\S` † = `\dag` ‡ = `\ddag` ¶ = `\P` © = `\copyright` £ = `\pounds`

The production of Greek letters and other mathematical symbols is described in Chapter 7.

2.5.6 Non-English letters

Special letters that exist in languages other than English can also be generated with TEX. These are:

```
œ={\oe}  Œ={\OE}  æ={\ae}  Æ={\AE}  å={\aa}  Å ={\AA}  ¡=!‘
ø ={\o}  Ø ={\O}  ł ={\l}  Ł ={\L}  ß={\ss}  SS={\SS}  ¿=?‘
```

Ångstrøm may be written as `{\AA}ngstr{\o}m` while *Karlstraße* can be input as `Karlstra{\ss}e`. The ‘letter’ `\SS` is the uppercase equivalent of `\ss`, used for automatic conversion between upper- and lowercase.

However, see Section 2.5.9 for the possibility of entering such characters directly.

2.5.7 Accents

In non-English languages, there is a multiplicity of *diacritical marks* or *accents*, most of which can be printed with TEX:

```
ò =\‘{o}   ó=\’{o}  ô=\^{o}  ö=\"{o}  õ=\~{o}
ō =\={o}   ȯ=\.{o}  ŏ=\u{o}  ǒ=\v{o}  ő=\H{o}
o͡o=\t{oo}  o̧=\c{o}  ọ=\d{o}  o̲=\b{o}  o̊=\r{o}
```

The *o* above is given merely as an example: any letter may be used. With *i* and *j* it should be pointed out that the dot must first be removed. This is carried out by prefixing these letters with a backslash: the commands `\i` and `\j` yield ı and ȷ. In this way ĭ and ǰ are formed by typing `\u{\i}` and `\H{\j}`.

The accent commands consisting of a non-letter may also be given without the curly braces:

```
ò=\‘o  ó=\’o  ô=\^o  ö=\"o  õ=\~o  ō=\=o  ȯ=\.o
```

The letter accent commands should always be used with the curly braces.

2.5.8 The euro symbol

The euro symbol € (or €) is too new to be part of the original LaTeX, but it can be produced with the help of some additional fonts and contributed packages. Just which package you may use depends on your installation and whether you have access to these additional fonts.

Package: `textcomp`

The *Text Companion* fonts, described in Section A.3.2, do contain a euro symbol. Since these fonts should be part of every modern LaTeX installation, you should be able to use their euro symbol if all else fails.

The package `textcomp` must be loaded in the preamble with

```
\usepackage{textcomp}
```

which defines many commands including `\texteuro` to print the symbol €. Since the European Commission originally dictated that it should only be printed in a sans serif font, it is better to issue `\textsf{\texteuro}` to produce €. (The font selection commands are described in Section 4.1.4.) If you are going to use this very frequently, you might want to define a shortcut named `\euro` with

```
\newcommand{\euro}{\textsf{\texteuro}}
```

as described in Section 10.3 on defining commands.

Package: `eurosym`

A better solution is presented by the `eurosym` package by Henrik Theiling and the associated fonts that come with it, which bear the names `feymr10`, `feybr10`, and so on. This package defines the `\euro` command to print €, which changes automatically to bold **€** and slanted *€* as needed.

Package: `europs`

The `europs` package by Joern Clausen interfaces to the type 1 (PostScript) euro fonts published by Adobe. For licensing reasons, these fonts may only be obtained from Adobe directly, though free of charge (see Section B.2). This package provides the command `\EUR` for a symbol that varies with font family (Roman €25, `sans serif` €25, and `typewriter` €25) as well as for bold **€25** and slanted *€25*. There is also a command `\EURofc` for the invariable symbol €.

Package: `eurosans`

Finally, the package `eurosans` by Walter Schmidt also addresses the Adobe euro fonts, again with the command `\euro`, with the same behavior as that of `eurosym`: always the sans serif family, but it changes with the other font attributes.

The following table summarizes the above packages:

Package	Command	Fonts	Notes
`textcomp`	`\texteuro`	Text Companion	Nonstandard symbol
`eurosym`	`\euro`	Eurosym	Sans serif, variable
`europs`	`\EUR`	PostScript	Varies with font family
	`\EURofc`		Invariable, official
`eurosans`	`\euro`	PostScript	Sans serif, variable

So which package should one use? That really depends on the fonts available. Since the Adobe fonts can never be distributed with a TEX installation, they must be actively fetched and installed. However, it is worth doing so, because the European Commission has revised its initial directive and now allows the euro symbol to be typographically matched to the text, which is also standard practice in Europe today. This strengthens the case for the `europs` package and the `\EUR` command for €, at least for Roman fonts.

2.5.9 Typing special symbols directly

The commands for producing the special characters and accented letters in the previous sections may be suitable for typing isolated 'foreign' words, but become quite tedious for inputting large amounts of text making regular use of such characters. Most computer systems provide non-English keyboards with appropriate fonts for typing these national variants directly. Unfortunately, the coding of such extra symbols is by no means standard, depending very much on the computer system.

For example, the text `Gauß meets Ampère` entered with an MS-DOS editor (using IBM code page 437 or 850) appears in a Windows application as `Gauá meets AmpŠre` and on a Macintosh as `Gau· meets Ampäre`. Since LATEX is intended to run on all systems, it simply ignores all such extra character codes on the grounds that they are not properly defined.

Package: `inputenc`

The `inputenc` package solves this problem. It not only informs LATEX which input coding scheme is being used, it also tells it what to do with the extra characters. One invokes it with

```
\usepackage[code]{inputenc}
```

where *code* is the name of the coding scheme to be used. The current list of allowed values for *code* (more are added with each LATEX update) can be found in Table D.1 on page 443. For most users, the most interesting codes are:

`cp437`	IBM code page 437 (DOS, North America)
`cp850`	IBM code page 850 (DOS, Western Europe)
`applemac`	Macintosh encoding
`ansinew`	Windows ANSI encoding

In short, you should select `applemac` for a Macintosh, and `ansinew` for Windows, and one of the others if you are working with DOS.

Documents making use of this package are fully portable to other computer systems. The source text produced with a DOS editor may still look very strange to a human user reading it on a Macintosh, but when the Macintosh LATEX processes it, the proper DOS interpretations will be applied so that the end result is what the author intended.

See Section D.4 for more details.

2.5.10 Ligatures

In book printing, certain combinations of letters are not printed as individuals but as a single symbol, called a *ligature*. TEX processes the letter combinations `ff`, `fi`, `fl`, `ffi`, and `ffl` not as

ff, fi, fl, ffi, ffl but rather as ff, fi, fl, ffi, ffl

Ligatures may be broken, that is, forced to be printed as separate letters, by inserting `\/` between the letters. This is sometimes desired for such words as *shelfful* (`shelf\/ful`), which looks rather strange when printed with the normal *ff* ligature, *shelfful*.

2.5.11 The date

The current date can be placed at any point in the text with the command `\today`. The standard form for the date is the American style of month, day, year (for example, July 15, 2003). The British form (15th July 2003) or the date in other languages can be generated with the help of the TEX commands `\day`, `\month`, and `\year`, which return the current values of these parameters as numbers. Examples of how such a new `\today` command may be made are shown on page 442 in Section D.3.2.

Package: `babel`

A more convenient way to do this is with the `babel` system for multilingual LATEX (Chapter 14), which automatically redefines `\today` for the current language or dialect. The British form is activated for the 'languages' `english`, `UKenglish`, and `canadian`, while the American style appears with the selections `american` and `USenglish`.

It is indeed better to enter the date explicitly, rather than to rely on `\today`. Reprocessing a two-year-old LATEX source file will yield a document with the current date, not the date when the text was written.

2.6 Exercises

Solutions to all the exercises in this book can be found on the enclosed TEX Live CD in `books→Guide4→exercises`.

Exercise 2.1: *This exercise tests the basic operations of running the LATEX program with a short piece of text. A few simple commands are also included. Use a text editor to produce the following source text and store it in a file named* `exer.tex`*.*

```
\documentclass{article}
\begin{document}
Today (\today) the rate of exchange between the British
pound and American dollar is \pounds 1 = \$1.71, an
increase of 1\% over yesterday.
\end{document}
```

Process this source file with LaTeX by clicking the appropriate icon, or by issuing `latex exer` *in a command window. If the processing occurs without any error messages, the* `.dvi` *file* `exer.dvi` *will have been successfully created and may be viewed by a dvi previewer or sent to a printer. The final printed result should look as follows except that your current date will appear:*

Today (July 15, 2003) the rate of exchange between the British pound and American dollar is £1 = $1.71, an increase of 1% over yesterday.

Note the following points about the commands used:

- *No blank is necessary after* `\today` *because the* `)` *suffices to terminate it.*
- *The blank after* `\pounds` *is optional, and it is not printed in the output.*
- *The commands* `\$` *and* `\%` *do not require blanks to terminate them; if blanks are given, they will be printed.*

Exercise 2.2: *Take some text of about 3/4 of a page long out of a book or journal article and type it into a LaTeX source file. Pay attention that the paragraphs are separated by blank lines. Use the same set of commands as in Exercise 2.1; that is, put the text between the commands* `\begin{document}...\end{document}` *and repeat the procedures for obtaining the output.*

Exercise 2.3: *If you are likely to need the euro symbol in your work, try redoing Exercise 2.1 as follows:*

```
\documentclass{article}
\usepackage{eurosym}
\begin{document}
Today (\today) the rate of exchange between the British
pound and European euro is \pounds 1 = \euro1.44, an
increase of 1\% over yesterday.
\end{document}
```

If this fails, try using one of the other packages described in Section 2.5.8, substituting `\textsf{\texteuro}` *or* `\EUR` *for* `\euro` *as required.*

2.7 Fine-tuning text

The subject of this section concerns pure typographical markup and has nothing to do with the logical markup that we wish to stress in this book. Unfortunately, there are times when the author or editor does have to help the typesetting program to achieve good appearance.

2.7.1 Word and character spacing

The spacing between words and characters is normally set automatically by TeX, which not only makes use of the natural width of the characters but also takes into account alterations for certain character combinations. For example, an A followed by a V does not appear as AV but rather as AV; that is, they are moved together

slightly for a more pleasing appearance. Interword spacing within one line is uniform and is chosen so that the right and left ends match exactly with the side margins. This is called left and right *justification*. TEX also attempts to keep the word spacing for different lines as nearly the same as possible.

TEX adheres to traditional typesetting rules according to which there should be extra spacing between sentences. Words that end with a punctuation mark are given extra spacing depending on the character: Following a period '.' or exclamation mark '!', there is more space than after a comma ','. In certain cases, the automatic procedures do not work properly, or it is desirable to override them, as described in the next sections. (Except in this paragraph and the next section, this book inserts no extra spacing between sentences.)

Sentence termination and periods

TEX interprets a period following a lowercase letter to be the end of a sentence where additional interword spacing is to be inserted. This leads to confusion with abbreviations such as *i. e.*, *Prof. Jones*, or *Phys. Rev.*, where the normal spacing is required. This can be achieved by using the characters ~ or \␣ instead of the normal blank. (The character ␣ is simply a symbol for the blank that is otherwise invisible.) Both these methods insert the normal interword spacing; in addition, ~ is a *protected space* that prevents the line from being broken at this point. The above examples should be typed in as `i.~e.`, `Prof.~Jones`, and `Phys.\ Rev.`, producing *i. e.*, *Prof. Jones*, and *Phys. Rev.* with the correct spacing and forcing the first two to be all on one line. In the third case, there is nothing wrong with putting *Phys.* and *Rev.* on different lines.

A period following an uppercase letter is not interpreted as the end of a sentence, but as an abbreviation. If it really is the end of a sentence, then it is necessary to add `\@` before the period in order to achieve the extra spacing. For example, this sentence ends with NASA. It is typed in as `This sentence ends with NASA\@.`

French spacing

The additional interword spacing between sentences can be switched off with the command `\frenchspacing`, which remains in effect until countermanded with `\nonfrenchspacing`. In this case, the command `\@` is ignored and may be omitted. This paragraph has been printed with `\frenchspacing` turned on so that all word spacings within one line are the same. It corresponds to the general usage in contemporary typesetting.

Character combinations "' and '"

A small spacing is produced with the command `\,`. This may be used, for example, to separate the double quotes " and " from the corresponding single quotes ' and ' when they appear together. For example, the text ``` ``\,`Beginning' and `End'\,'' ``` produces "'Beginning' and 'End'".

Inserting arbitrary spacing

Spacing of any desired size may be inserted into the text with the commands

```
\hspace{space}
\hspace*{space}
```

where *space* is the length specification for the amount of spacing, for example, `1.5cm` or `3em`. (Recall that one *em* is the width of the letter M in the current typeface.)

This command puts blank space of width *space* at that point in the text where it appears. The standard form (without *) has no effect if it should come at the beginning of an output line, just as normal blanks are removed at the beginning of lines. The *-form, on the other hand, inserts the spacing no matter where it occurs.

A blank before or after the command will also be included:

```
This is\hspace{1cm}1cm          This is 1cm
This is \hspace{1cm}1cm         This is  1cm
This is \hspace{1cm} 1cm        This is  1cm
```

The length specification may be negative, in which case the command works as a backspace for overprinting characters with other ones or moving them closer together. For example, there is an energy unit in physics called *electron volt*, abbreviated 'eV', which looks much better if the two letters are nearer together, as 'eV', with `e\hspace{-.12ex}V`.

The command `\hfill` is an abbreviation for `\hspace{\fill}` (see Section 2.4.2). It inserts enough space at that point to force the text on either side to be pushed over to the left and right margins. With `Left\hfill Right` one produces

Left Right

Multiple occurrences of `\hfill` within one line will each insert the same amount of spacing so that the line becomes left and right justified. For example, the text `Left\hfill Center\hfill Right` generates

Left Center Right

If `\hfill` comes at the beginning of a line, the spacing is suppressed in accordance with the behavior of the standard form for `\hspace`. If a rubber space is really to be added at the beginning or end of a line, `\hspace*{\fill}` must be used instead. However, LaTeX also offers a number of commands and environments to simplify most such applications (see Section 4.2.2).

A number of other fixed horizontal spacing commands are available:

`\quad` and `\qquad`

The command `\quad` inserts a horizontal space equal to the current type size, that is, 10 pt for a 10 pt typeface, whereas `\qquad` inserts twice as much.

Inserting variable ..... and _____ sequences

Two commands that work exactly the same way as `\hfill` are

`\dotfill` and `\hrulefill`

Instead of inserting empty space, these commands fill the gap with dots or a ruled line, as follows:

```
Start \dotfill\ Finish\\                        and
Left \hrulefill\ Center \hrulefill\ Right\\     produce
```

Start ..... Finish
Left _____ Center _____ Right

Any combination of `\hfill`, `\dotfill`, and `\hrulefill` may be given on one line. If any of these commands appears more than once at one location, the corresponding filling will be printed that many more times than for a single occurrence.

```
Departure \dotfill\dotfill\dotfill\ 8:30 \hfill\hfill
Arrival \hrulefill\ 11:45\\
```

Departure .... 8:30 Arrival _____ 11:45

2.7.2 Line breaking

Breaking text into lines is done automatically in TEX and LATEX. However, there are times when a line break must be forced or encouraged, or when a line break is to be suppressed.

The command \\

A new line with or without additional line spacing can be achieved with the command `\\`. Its syntax is

`\\[`*space*`]`
`\\*[`*space*`]`

The optional argument *space* is a length that specifies how much additional line spacing is to be put between the lines. If it is necessary to start a new page, the additional line spacing is not included and the new page begins with the next line of text. The *-form prevents a new page from occurring between the two lines.

With `\\*[10cm]`, the current line is ended and a vertical spacing of 10 cm is inserted before the next line, which is forced to be on the same page as the current line. If a page break is necessary, it will be made before the current line, which is then positioned at the top of the new page together with the 10 cm vertical spacing and the next text line.

The command `\newline` is identical to `\\` without the option *space*. That is, a new line is started with no additional spacing and a page break is possible at that point.

Both commands may be given only within a paragraph, and not between them where they would be meaningless.

Further line-breaking commands

The command \linebreak is used to encourage or force a line break at a certain point in the text. Its form is

\linebreak[*num*]

where *num* is an optional argument, a number between 0 and 4 that specifies how important a line break is. The command recommends a line break, and the higher the number the stronger the recommendation. A value of 0 allows a break where it otherwise would not occur (such as in the middle of a word), whereas 4 compels a line break, as does \linebreak without *num*. The difference between this command and \\ or \newline is that the current line will be fully justified; that is, interword spacing will be added so that the text fills the line completely. With \\ and \newline, however, the line is filled with empty space after the last word and the interword spacing remains normal.

The opposite command

\nolinebreak[*num*]

discourages a line break at the given position, with *num* specifying the degree of discouragement. Again, \nolinebreak without a *num* argument has the same effect as \nolinebreak[4]; that is, a line break is absolutely impossible here.

Another way of forcing text to stay together on one line is with the command \mbox{*text*}. This is convenient for expressions such as 'Voyager-1' to stop a line break at the hyphen.

2.7.3 Vertical spacing

It is possible to add extra vertical spacing of amount *space* between particular paragraphs using the commands

\vspace{*space*}
\vspace*{*space*}

The *-form will add the extra space even when a new page occurs or when the command appears at the top of a new page. The standard form ignores the extra vertical spacing in these situations.

If these commands are given within a paragraph, the extra space is inserted after the current line, which is right and left justified as usual.

The *space* parameter may even be negative in order to move the following text higher up on the page than where it would normally be printed.

The command \vfill is an abbreviation for \vspace{\fill} (see Section 2.4.2). This is the equivalent of \hfill for vertical spacing, inserting enough blank vertical space to make the top and bottom of the text match up exactly with the upper and lower margins. The comments on multiple occurrences of \hfill also apply to \vfill. If this command is given at the beginning of a page, it is ignored, just like

the standard form of \vspace{\fill}. If a rubber space is to be put at the top of a page, the *-form \vspace*{\fill} must be used.

Further commands for increasing the spacing between paragraphs are

```
\bigskip    \medskip    \smallskip
```

which add vertical spacing depending on the font size declared in the document class.

2.7.4 Page breaking

Breaking text into pages occurs automatically in TEX and LATEX, just as for line breaking. Here again, it may be necessary to interfere with the program's notion of where a break should take place.

Normal pages

The commands

```
\pagebreak[num]
\nopagebreak[num]
```

are the equivalents of \linebreak and \nolinebreak for page breaking. If the command \pagebreak appears between two paragraphs, a new page will be forced at that point. If it comes within a paragraph, the new page will be implemented after the current line is completed. This line will be right and left justified as usual.

The command \nopagebreak has the opposite effect: Between paragraphs it prevents a page break from occurring there, and within a paragraph it stops a page break that might take place at the end of the current line.

Optional numbers between 0 and 4 express the degree of encouragement or discouragement for a page break. The analogy with the command \linebreak goes further: Just as the line before the break is left and right justified with extra interword spacing, the page before the break is expanded with interline spacing to make it top and bottom justified.

The proper command to end a page in the middle, fill it with blank spacing, and go on to a new page is

```
\newpage
```

which is equivalent to \newline with regard to page breaking.

Pages with figures and tables

If the text contains tables, pictures, or reserved space for figures, these are inserted at the location of the corresponding command provided that there is enough room for them on the current page. If there is not enough space, the text continues and the figure or table is stored to be put on a following page.

The command

```
\clearpage
```

ends the current page like `\newpage` and, in addition, outputs all the pending figures and tables on one or more extra pages (Chapter 9).

Two-column pages

If the document class option `twocolumn` has been selected, or the command `\twocolumn` has been issued, then the two commands `\pagebreak` and `\newpage` end the current *column* and begin a new one, treating columns as pages. On the other hand, `\clearpage` and `\cleardoublepage` (see below) terminate the current page, inserting an empty right column if necessary.

Two-sided pages

An additional page-breaking command is available when the document class option `twoside` has been selected:

```
\cleardoublepage
```

It functions exactly the same as `\clearpage` (the current page is terminated, all pending figures and tables are output) but, in addition, the next text will be put onto an *odd*-numbered page. If necessary, an empty page with an even number is printed to achieve this.

Controlling page breaks

LaTeX provides the possibility of increasing the height of the current page slightly with commands

```
\enlargethispage{size}
\enlargethispage*{size}
```

which add the length *size* to `\textheight` for this one page only. Sometimes the difference of a few points is all that is necessary to avoid a bad page break. The *-form of the command also shrinks any interline spacing as needed to maximize the amount of text on the page.

2.8 Word division

When a line is to be right and left justified, it often happens that the break cannot be made between whole words without either shoving the text too close together or inserting huge gaps between the words. It is then necessary to split a word. This fundamental task is performed by TeX, the underlying basis of LaTeX, by means of a word-dividing algorithm that works (almost) perfectly for English text, which is more than can be said for most authors. Nevertheless, even it makes mistakes at times, which need to be corrected by human intervention.

If normal TEX/LATEX is used for other languages, or if foreign words appear in English text, incorrect hyphenations are very likely to appear. (See Section 2.8.4 and Chapter 14 for more about LATEX with other languages.) In these cases too, something must be done to override TEX's hyphenation rules, as described below.

2.8.1 Manual hyphenation

The simplest way to correct a wrongly divided word is to include a `\-` command at the right place within the word. The word *manuscript*, for example, will not be hyphenated at all, so if it causes problems with breaking a line, write it as `man\-u\-script`. This tells TEX to divide the word as necessary either as *man-uscript* or as *manu-script*, and to ignore its normal rules.

The `\-` command merely makes hyphenation possible at the indicated locations; it does not force it. If the author absolutely insists on dividing a word at a certain point, say between the *u* and *s* in *manuscript*, he or she can type `manu-\linebreak script` to achieve this. However, this brute-force method is not recommended, because the line break will always occur here even if the text is later changed.

For English text, the spelling of a word remains the same when it is hyphenated, something that is not true in other languages. In traditional German spelling, for example, if *ck* is split, it becomes *k-k*. TEX allows such behavior with the general hyphenation command

`\discretionary{`*before*`}{`*after*`}{`*without*`}`

where *before* and *after* are the letters (with hyphen) that come on either side of the break if division takes place, and *without* is the normal text with no hyphenation. Thus Boris Becker's name should be typed as

```
Boris Be\discretionary{k-}{k}{ck}er
```

something that one only wants to do in exceptional situations. Incidentally, the `\-` command is shorthand for `\discretionary{-}{}{}`.

Note: In today's new German spelling, *ck* is never split. This reformed spelling is still controversial, however.

2.8.2 Hyphenation list

Words that are incorrectly hyphenated and that appear frequently within the document can be put into a *list of exceptions* in the preamble, to avoid laboriously inserting `\-` every time:

`\hyphenation{`*list*`}`

The *list* consists of a set of words, separated by blanks or new lines, with the allowed division points indicated by hyphens. For example,

```
\hyphenation{man-u-script com-pu-ter gym-na-sium
             coun-try-man re-sus-ci-tate ... }
```

The list may contain only words with the normal letters *a–z*, with no special characters or accents. However, if the `inputenc` package is loaded, then the directly typed letters are also included in the automatic hyphenation.

2.8.3 Suppressing hyphenation

Another means of avoiding bad word divisions is to turn hyphenation off, at least for a paragraph or two. Actually, the environment

`\begin{sloppypar}` *paragraph text* `\end{sloppypar}`

does not prevent word division, but it does permit larger interword spacings without giving a warning message. This means that practically all lines are broken between words. It is also possible to put the command `\sloppy` in the preamble or in the current environment to reduce the number of word divisions in the whole document or within the environment scope. This is recommended when the line width is rather narrow.

When the command `\sloppy` is in effect, it is possible to undo it temporarily and to turn hyphenation back on with the command `\fussy`.

2.8.4 Word division with multilingual text

Multiple hyphenation lists may be included in the TEX format, making it possible to switch hyphenation schemes within one document, using the TEX command `\language`. This command may be used as part of language-specific adaptations to translate certain explicit English words in the output (such as ‘Contents’), to simplify accents or punctuation, and to alter the definition of the date command `\today`. This topic is treated in more detail in Chapter 14.

This page intentionally left blank

3 Document Layout and Organization

3.1 Document class

The first command in the preamble of a LaTeX file determines the global processing format for the entire document. Its syntax is:

```
\documentclass[options]{class}
```

where some value of *class* must be given, while [*options*] may be omitted if default values are acceptable.

Class: `article` `report` `book`

The standard values of *class*, of which one and only one may be given, are `book`, `report`, `article`, or `letter`. (The properties of the `letter` class are explained in Chapter 18.) The basic differences between these classes lie not only in the page layouts, but also in the organization. An article may contain *parts*, *sections*, *subsections*, and so on, while a report can also have *chapters*. A book also has chapters, but treats even and odd pages differently; also, it prints running heads on each page with the chapter and section titles.

Other classes besides the standard ones exist, as contributions for specific journals or for book projects. These will have their own set of *options* and additional commands, which should be described in separate documentation or instructions. However, since they are normally modifications of one of the standard classes, most of the following options apply to them too.

3.1.1 Standard class options

The *options* available allow various modifications to be made to the formatting. They can be grouped as follows.

Selecting font size

The basic font size is selected with one of the options

```
10pt    11pt    12pt
```

This is the size of the font in which the normal text in the document will be set. The default is `10pt`, which means that this is the value assumed if no size option is specified. All other font-size declarations are relative to this standard size so that the section titles, footnotes, and so on will all change size automatically if a different basic font size is selected.

Specifying paper size

LaTeX calculates the text line width and lines per page according to the selected font size and paper mode. It also sets the margins so that the text is centered both horizontally and vertically. To do this, it needs to know which paper format is being used. This is specified by one of the following options:

`letterpaper` (11×8.5 in)	`a4paper` (29.7×21 cm)
`legalpaper` (14×8.5 in)	`a5paper` (21×14.8 cm)
`executivepaper` (10.5×7.25 in)	`b5paper` (25×17.6 cm)

The default is `letterpaper`, American letter-size paper, 11×8.5 in.

Normally, the paper format is such that the longer dimension is the vertical one, the so-called *portrait* mode. With the option

```
landscape
```

the shorter dimension becomes the vertical one, the *landscape* mode. (One still has to ensure that the output is printed as landscape; see, for example, page 242.)

Page formats

The text on the page may be formatted into one or two columns with the options

```
onecolumn    twocolumn
```

The default is `onecolumn`. In the case of the `twocolumn` option, the separation between the columns as well as the width of any rule between them may be specified by `\columnsep` and `\columnseprule`, described below.

The even- and odd-numbered pages may be printed differently according to the options

```
oneside    twoside
```

With `oneside`, all pages are printed the same; however, with `twoside`, the running heads are such that the page number appears on the right on odd pages and on the left on even pages. *It does not force the printer to output double-sided.* The idea is that when these are later printed back to back, the page numbers are always on the outside where they are more easily noticed. This is the default for the `book` class. For `article` and `report`, the default is `oneside`.

With the `book` class, chapters normally start on a right-hand, odd-numbered page. The options

`openright` `openany`

control this feature: With `openany`, a chapter always starts on the next page, but with `openright`, the default, a blank page may be inserted if necessary.

Normally the title of a `book` or `report` will go on a separate page, while for an `article` it is placed on the same page as the first text. With the options

`notitlepage` `titlepage`

this standard behavior may be overruled. See Sections 3.3.1 and 3.3.2.

Further options

The remaining standard options are:

`leqno` Equation numbers in displayed formulas will appear on the left instead of the normal right side (Section 7.1).

`fleqn` Displayed formulas will be set flush left instead of centered (Section 7.1). The amount of indentation may be set with the parameter `\mathindent` described below.

`openbib`
The format of bibliographies may be changed so that segments are set on new lines. By default, the texts for each entry are run together.

`draft` If the LaTeX line-breaking mechanism does not function properly and text must stick out into the right margin, then this is marked with a thick black bar to make it noticeable.

`final` The opposite of `draft` and the default. Lines of text that are too wide are not marked in any way.

If multiple options are to be given, they are separated by commas, as for example, `\documentclass[11pt,twoside,fleqn]{article}`. The order of the options is unimportant. If two conflicting options are specified, say `oneside` and `twoside`, it is not obvious which one will be effective. That depends entirely on the definitions in the class file itself, so it would be best to avoid such situations.

Parameters associated with some options

Some options make use of parameters that have been given certain default values:

`\mathindent`
specifies the indentation from the left margin for the equation numbers when `fleqn` is selected (Section 7.1);

`\columnsep`
specifies the space between the two columns for the `twocolumn` option (see Figure G.2 on page 555);

`\columnseprule`
: determines the width of the vertical line between the two columns for the `twocolumn` option. The default is zero width; that is, no vertical rule (see Figure G.2).

The standard values of these parameters may be changed with the LaTeX command `\setlength`. For example, to change `\mathindent` to 2.5 cm, give

```
\setlength{\mathindent}{2.5cm}
```

These parameters may be assigned values either in the preamble or at any place in the document. Parameters in the preamble apply to the entire document, whereas those within the text are in effect until the next change or until the end of the environment in which they were made (Section 2.3). In the latter case, the previous values become effective once more.

Exercise 3.1: *Take your text file from Exercise 2.2 and change the initial command `\documentclass{article}` first to `\documentclass[11pt]{article}` and then to `\documentclass[12pt]{article}` and print the results of each LaTeX processing. Compare the line breaking of these outputs with that of Exercise 2.2.*
Note: If there are some improper word divisions, you can tell LaTeX where the correct division should occur with the command `\-`, for example, `man\-u\-script`. (This is one of the few words that the TeX English word divider does not handle properly.) Additional means of modifying word division are given in Section 2.8.
If there are warnings of the sort `Overfull \hbox ...` during the LaTeX processing, TeX was not able to break the lines cleanly. In the output, these lines will extend beyond the right margin. The usual cause is that TeX was not able to divide some word, either because it is indivisible or because TeX's word division routines were not adequate. Here again a suggested hyphenation in the text can solve the problem. Other solutions will be given shortly.

Exercise 3.2: *Now employ `\documentclass[twocolumn]{article}` in your text file. If you now receive a number of warnings with `Underfull \hbox ...`, then these lines will indeed be left and right justified but will have too much empty space between the words. Check the output yourself to see whether the word spacing is acceptable. If not, try giving some hyphenation suggestions in the first words of the next line.*
Note: If you use the classes `book` or `report` instead of `article` in the preceding exercises, you will notice no difference in the outputs. These classes affect the subsequent structural elements of the document. Basically, you should use `article` for short articles (say, 10–20 pages) and `report` for longer reports that are to be organized into chapters. The chapters always begin on a new page. The class `book` is available for producing books.

3.1.2 Loading packages

In Section 1.5.3 we explained how LaTeX can be extended by *packages*, which are either part of the core installation or contributed by engaged users. How to write your own packages is described in Appendix D.

A package is nothing more than a set of LaTeX (or TeX) commands stored in a file with the extension `.sty`, although there are some special commands that may only appear within them. To invoke a package, simply call

```
\usepackage{package}
```

in the preamble, where *package* is the root name of the file. More than one package may be loaded with one call to `\usepackage`. For example, two packages provided with standard LaTeX are stored in files `makeidx.sty` (Section 11.4.3) and `ifthen.sty` (Section 10.3.5). They may be read in together with

```
\usepackage{makeidx,ifthen}
```

A package may have options associated with it, which may be selected in the same way as for document classes: by including the option names within square braces. The general syntax is:

```
\usepackage[opt1,opt2...]{package1,package2,...}
```

where all the listed options will be applied to all the selected packages. If any of the packages does not understand one of the options, a warning message is output to the monitor.

3.1.3 Global and local options

One interesting feature about options specified with the `\documentclass` command is that they also apply to any packages that follow. This means that if several packages take the same option, it is only necessary to declare it once in `\documentclass`. For example, one might design a package to modify `article` for generating a local house style that might do different things for single- or double-column text; this package could make use of the class options `onecolumn` and `twocolumn` to achieve this. Or, it could elaborate on the `draft` option to produce double line spacing, as for a manuscript. Alternatively, several packages might have language-dependent features that could be activated with options like `french` or `german`; it is sufficient to list such options only in `\documentclass` to apply them to all packages. Such options are called *global*, for they are passed on to all subsequent packages automatically.

Global options need not be limited to the standard class options listed in Section 3.1.1. A warning message is printed only if neither the class nor any of the packages understand one or more of them. By contrast, any options specified with `\usepackage` will be applied only to those packages listed in that one command, and it is applied to all of them. A warning is printed if one or more of those packages does not recognize any one of these *local* options.

3.1.4 Class and package versions

Class and package files normally have an internal version specification in the form of their release date, as *yyyy/mm/dd*. If you wish to make use of some feature that you know was added on a certain date, you include that date in square brackets after the class or package name. An example of this is shown in Section 3.2.4 on page 50.

The version date may also be added to the `\documentclass` command to ensure that the right version of the class file is being employed. The reason for doing this is to ensure that the source files are processed properly, say, on other systems.

3.2 Page style

The basic page format is determined by the *page style*. With one exception, this command is normally given in the preamble. Its form is:

\pagestyle{*style*}

The mandatory argument *style* takes on one of the following values:

`plain` The page head is empty; the foot contains the centered page number. This is the default for the `article` and `report` classes when no `\pagestyle` is given in the preamble.

`empty` Both head and footlines are empty; no page numbers are printed.

`headings`
The head contains the page number as well as title information (chapter and section headings); the foot is empty. This is the default for `book` class.

`myheadings`
The same as `headings` except that the page titles in the head are not chosen automatically but rather are given explicitly by the commands `\markright` or `\markboth` (see below).

The command

\thispagestyle{*style*}

functions exactly as `\pagestyle` except that it affects only the current page. For example, the page numbering may be suppressed for just the current page with the command `\thispagestyle{empty}`. It is only the *printing* of the page number that is suppressed; the next page will be numbered just as though the command had never been given.

3.2.1 Heading declarations

For the page styles `headings` and `myheadings`, the information appearing in the headline may be given with the declarations

\markright{*right_head*}
\markboth{*left_head*}{*right_head*}

The declaration `\markboth` is used with the document class option `twoside`, with even-numbered pages considered to be on the *left* and odd-numbered pages on the *right*. Furthermore, the page number is printed on the left side of the head for a left page and on the right side for a right page.

For one-sided output, all pages are considered to be right-handed. In this case, the declaration `\markright` is appropriate. It may also be used with two-sided output to overwrite the *right_head* given in `\markboth`.

With the page style `headings`, the standard titles in the page headline are the chapter, section, or subsection headings, depending on the document and page style, according to the following scheme:

Style		Left Page	Right Page
book, report	one-sided	—	*Chapter*
	two-sided	*Chapter*	*Section*
article	one-sided	—	*Section*
	two-sided	*Section*	*Subsection*

If there is more than one \section or \subsection on a page, the heading of the last one appears in the page head.

3.2.2 Customized head and footlines

Package: fancyhdr

The standard page styles described in Section 3.2 select how the head and footlines are to appear, and what information they contain. This is a very limited choice, and the fancyhdr package by Piet van Oostrum offers the user considerably more flexibility.

This package makes available an additional page style named fancy that the user can easily redefine. Head and footlines consist each of three parts—left, center, and right—each of which can be individually defined with

\lhead{*Left head*} \chead{*Center head*} \rhead{*Right head*}
\lfoot{*Left foot*} \cfoot{*Center foot*} \rfoot{*Right foot*}

where the various texts may be explicit, or a command such as \thepage to print the current page number. Both head and footlines may be decorated with a rule, the widths of which are set by commands \headrulewidth and \footrulewidth. By default, the fancy head and footlines are much the same as for the headings page style, but the head rule is set to 0.4 pt and the foot rule set to 0 (no rule). The rules may be redefined with, for example,

```
\renewcommand{\footrulewidth}{0.4pt}
```

to turn on the foot rule.

The above defining commands are in fact specific examples of the more general commands \fancyhead and \fancyfoot, where

\lhead{..} is \fancyhead[L]{..}
\cfoot{..} is \fancyfoot[C]{..}

and so on, with L C R standing for 'left', 'center', 'right'.

For two-sided output with the twoside option, one normally wants the left and right parts to alternate with the page number. The easiest way to do this is with

\fancyhead[LE,RO]{*Text 1*} \fancyhead[LO,RE]{*Text 2*}

to put the same *Text 1* in the left part of even pages, and right part of odd pages, and *Text 2* for the other way around. With \fancyhead{}, all headline parts are set to blanks, something that should be done before resetting them explicitly. Similarly, \fancyfoot{} sets all foot entries to blank.

The default (two-sided) definitions for the fancy page style are

```
\fancyhead[EL,OR]{\textsl{\rightmark}}
\fancyhead[ER,OL]{\textsl{\leftmark}}
```

where `\rightmark` and `\leftmark` contain the automatic texts for the `headings` page style generated by the `\chapter`, `\section`, `\subsection` commands (Section 3.3.3), while `\textsl` (Section 4.1.4) sets its argument in a slanted typeface. The user may also make use of these to redefine the headline with automatic texts.

There is also the most general `\fancyhf` command taking optional arguments `[H]` and `[F]` to apply to head or footlines. Thus `\fancyhf[HL]{..}` is the same as `\fancyhead[L]{..}`. Clearly, `\fancyhf{}` resets everything.

In many classes, the first page of a chapter, or the very first page of the document, is switched to `plain` automatically. If the user wants to change this, he or she must redefine that page style. The `fancyhdr` package simplifies this task with

`\fancypagestyle{plain}{`*definitions*`}`

where *definitions* consist of `\fancyhead`, `\fancyfoot`, and/or rule redefinitions that are to apply to the revised `plain` style. In fact, any existing page style can be redefined in this way.

3.2.3 Page numbering

The declaration that specifies the style of the page numbering has the form

`\pagenumbering{`*num_style*`}`

The allowed values of *num_style* are:

`arabic`	for normal (Arabic) numerals,
`roman`	for lowercase Roman numerals,
`Roman`	for uppercase Roman numerals,
`alph`	for lowercase letters,
`Alph`	for uppercase letters.

The standard value is `arabic`. This declaration resets the page counter to 1. To paginate the foreword of a document with Roman numerals and the rest with Arabic numbers beginning with page 1 for chapter 1, one must declare `\pagenumbering{roman}` at the start of the foreword and then reset the page numbering with `\pagenumbering{arabic}` immediately after the first `\chapter` command. (See Section 3.3.5 for the preferred method.)

Pages may be numbered starting with a value different from 1 by giving the command

`\setcounter{page}{`*page_num*`}`

where *page_num* is the number to appear on the current page.

Exercise 3.3: *Expand your exercise text file so that it fills more than one page of output and include the following preamble:*

```
\documentclass{article}
\pagestyle{myheadings} \markright{Exercises}
\pagenumbering{Roman}
\begin{document}
```

3.2.4 Paragraph formatting

The following parameters affect the appearance of a paragraph and may be given new values with `\setlength` as explained in Section 10.2:

`\parskip`
: The distance between paragraphs, expressed in units of `ex` so that it will automatically change with character font size. This should be a *rubber* length.

`\parindent`
: The amount of indentation for the first line of a paragraph.

`\baselinestretch`
: This is a number that magnifies the normal distance between *baselines*, the line on which the letters sit. This number is initially 1, for standard line spacing. It may be changed to another number with

  `\renewcommand{\baselinestretch}{`*factor*`}`

  where *factor* is any decimal number, such as 1.5 for a 50% increase. This then applies to all font sizes. If this command is given outside the preamble, it does not come into effect until another font size has been selected (Section 4.1.2).

These parameters may be set either in the preamble or anywhere in the text of the document. In the latter case, the changes remain in effect until the next change or until the end of the environment in which they were made (Section 2.3).

To suppress indentation for one paragraph, or to force it where it would otherwise not occur, place

`\noindent` or `\indent`

at the beginning of the paragraph to be affected.

Package: `indentfirst`

Normally, the first paragraph of a section is not indented, not even with `\indent`. However, by including the package `indentfirst` one ensures that all paragraphs are indented.

Package: `parskip`

By default, LaTeX indicates paragraphs by indenting the first line. An alternative is without indentation but with extra spacing between paragraphs. One could redefine `\parindent` and `\parskip` accordingly, or one could employ one of the oldest and simplest packages dating back to LaTeX 2.09 days: `parskip`, written by H. Partl. For consistency, this package also makes some changes in the parameters for lists (Section 4.3). One loads this package with