LaTeX for Complete Novices

Nicola Talbot

Monday 27th September, 2004

Contents

List of Figures

List of Tables

List of Exercises

Chapter 1

Introduction

The aim of this document is to introduce LaTeX to a non-technical person. LaTeX is excellent for producing professional looking documents, however it is a *language* not a word processor, so it can take a bit of getting used to, particularly if you have never had any experience using programming languages.

LaTeX does take a while to learn, so why should I use it? Firstly, LaTeX is far better at typesetting mathematical equations than word processors. Compare the following equations:

1. Using equation editor in Microsoft Word[1]:

$$\frac{\partial^2 L}{\partial z_i^{\rho\,2}} = -\frac{\partial \rho_i}{\partial z_i^{\rho}}\left(\frac{\partial v_i}{\partial \rho_i}\frac{e^{v_i}}{1-e^{v_i}} + v_i\frac{e^{v_i}\frac{\partial v_i}{\partial \rho_i}(1-e^{v_i}) + e^{2v_i}\frac{\partial v_i}{\partial \rho_i}}{(1-e^{v_i})^2}\right)$$

2. Using LaTeX:

$$\frac{\partial^2 \mathcal{L}}{\partial z_i^{\rho\,2}} = -\frac{\partial \rho_i}{\partial z_i^{\rho}}\left(\frac{\partial v_i}{\partial \rho_i}\frac{e^{v_i}}{1-e^{v_i}} + v_i\frac{e^{v_i}\frac{\partial v_i}{\partial \rho_i}(1-e^{v_i}) + e^{2v_i}\frac{\partial v_i}{\partial \rho_i}}{(1-e^{v_i})^2}\right)$$

(Incidentally, this equation was taken from some kernal survival analysis, so it is a genuine piece of mathematics. You will find out how to create this equation on page 104 in Section 9.3.8.)

Secondly, LaTeX makes it very easy to cross-reference chapters, sections, equations, figures, tables etc, and it also makes it very easy to generate a table of contents, list of figures, list of tables, index, glossary and bibliography. You don't need to worry about numbering anything, as this is done automatically, which means that you can insert new sections or swap sections around without having to worry about updating all the section numbering etc. LaTeX can also ensure consistent formatting, and the style of the document can be completely changed simply by using a different class file, or loading additional packages.

Thirdly, when you are editing a document using a word processor, the word processor has to work out how to reformat the document everytime you type something. If you have a large document with a great many inserted objects (such as figures and equations), the response to keyboard input can become very slow. You may find that after typing a few words you will have to wait until the computer

[1] I was unable to find a caligraphic font for the $\mathcal{L}$. The font looks a little ragged because I had to convert it to bitmap to include it into the document.

catches up before you can see what you have typed. With LaTeX you type your code in using an ordinary text editor. The document doesn't get formatted until you pass it to LaTeX, which means that you are not slowed down by constant reformatting.

Lastly, there's the fact that LaTeX follows certain typographical rules, so you can leave most of the typesetting to LaTeX. You rarely need to worry about minor things such as remembering to put two spaces between sentences and only one space between words, as LaTeX will do this automatically, and it will also automatically deal with f-ligatures. That is, if any of the following combination of letters are found: `fl`, `ffl`, `ff`, `fi`, `ffi`, they will automatically be converted into the corresponding ligatures: fl, ffl, ff, fi, ffi. Note the difference between fluffier (2 ligatures) and fluffier (no ligatures). These points may seem minor but they all contribute towards the impact of the entire document. When writing technical documents, the presentation as well as the content is important. All too often examiners, referees etc are put off reading a document because it is badly formatted. This provokes an immediate negative reaction and provides little desire to look favourably upon your work.

To give you an idea of what you can do with LaTeX, this document was written in LaTeX. The PostScript version was generated using LaTeX, `makeindex` and `dvips`, the PDF versions were generated using PDFLaTeX and `makeindex` and the HTML version was generated using the LaTeX2HTML[2] converter. All versions were generated from the same source code with occasional switches for minor variations between formats[3].

This document is structured as follows: Chapter 2 defines terms that will be used throughout this document. If you like, you can give this chapter a cursory glance to begin with and go back to it later. Chapter 3 details the software that you will need to use LaTeX and describes how to use the software. Chapter 4 shows you how to create a very basic document. Chapter 5 shows you how to create chapters, sections etc so that you end up with a fully structured document. Chapter 6 shows you how to load packages, and also how to download and install additional packages that weren't installed with your LaTeX distribution. Chapter 7 describes how to create figures and tables. Chapter 8 describes how to define your own commands. Chapter 9 describes how to typeset mathematics. Chapter 10 describes how to define new environments. Chapter 11 discusses counters. Chapter 12 discusses lengths, and Chapter 13 documents possible errors you may encounter, and gives advice on how to fix them.

This document and associated files are available on-line at: `http://theoval.cmp.uea.ac.uk/~nlct/latex/novices/`. This document is also available in 6×4in PDF format for on-line viewing and HTML format. If you are viewing this document in `Acrobat Reader`, you can click on the bookmarks tab to help navigate your way around the document.

1.1 Recommended Reading

This document is designed as an introductory text, not a comprehensive guide. For further reading try some of the following:

"LaTeX : a document preparation system" by Leslie Lamport [1] is the user's guide and reference manual for LaTeX, and is a good basic text for anyone starting out, however it doesn't cover AMSTeX, so anyone who needs to typeset more than basic mathematics may prefer either "A guide to LaTeX" by Helmut Kopka and Patrick Daly [2] or "The LaTeX companion" by Michel Goossens, Frank Mittelbach and Alexander Samarin [3]. Both these books cover AMSTeX, BibTeX and

[2] `http://www.latex2html.org/`

[3] plus a small Perl script to generate the file size information that appears at the start of the HTML version

`makeindex`. "A guide to LaTeX" also has an appendix that contains a brief summary of all commands described in the book for a quick and easy reference which is quite useful.

In the same series as "The LaTeX companion", there is also "The LaTeX graphics companion" by Michel Goossens, Sebastian Rahtz and Frank Mittelbach [4] which details how to illustrate documents with LaTeX and PostScript, including a chapter on colour (coloured text, background, tables and slides). This is recommended to anyone who is contemplating heavy use of graphics, but you do need a basic knowledge of LaTeX before delving into it.

The final book in the "Companion" series is "The LaTeX web companion" by Michel Goossens, Sebastian Rahtz *et al.* [5] which is recommended for those interested in creating documents for the web, either as HTML or PDF. This book details how to convert LaTeX documents into HTML using various applications such as `LaTeX2HTML` and `TeX4ht`, and how to create PDF documents using `PDFLaTeX`, including how to create active links within your document using the `hyperref` package.

There is also a wealth of LaTeX-related information on the world wide web. The Comprehensive TeX Archive Network[4] (CTAN) is a good place to start. In the UK, the TeX Archive at `http://www.tex.ac.uk/` is closer. You can check the on-line catalogue for information about available software, and there is also a list of frequently asked questions which I recommend you try if you have any queries. You can also try using a search engine, such as Google, but take care not to simply search for "latex" or you will end up with thousands of hits, most of which will be totally irrelevant. Search engines are unable to tell the difference between LaTeX (the typesetting language) and latex (the plant substance or synthetic product), so I would recommend that you use the advance search facility to exclude certain obvious words related to the latter (and you might also want to consider selecting the "filter using SafeSearch" option.) It would also be a good idea to specify a few extra words to help narrow down your search. For example, if you want a general introduction to LaTeX, you could type `latex` into the box marked "with **all** of the words" and type `introduction beginners guide novices` into the box marked "with **at least one** of the words". Alternatively, if you have an error message you don't understand, you could try typing part of the error message into the box marked "with the **exact phrase**".

[4] http://www.ctan.org/

Chapter 2

Some Definitions

As mentioned in the previous chapter, LaTeX is a language, so you can't simply start typing and expect to see your document appear before your very eyes. You need to know a few things before you can get started, so it's best to define a few terms first. Don't worry if there seems a lot to take in, there will be some practical examples later, which should hopefully make things a little clearer.

Throughout this document, source code is illustrated by a typewriter font with the word Input placed in the margin, and the corresponding output is typeset with the word Output in the margin. For example:

Sample Code:

```
This is an \textbf{example}.
```
Input

Resulting output:

This is an **example**.
Output

Segments of code that are longer than one line are bounded above and below by a horizontal line, illustrated as follows:

↑Input
```
Line one\par
Line two\par
Line three.
```
↓Input

with corresponding output:

↑Output

Line one
Line two
Line three.

↓Output

Command definitions are shown in a typewriter font in the form:

`\documentclass[`*options*`]{`*class file*`}`
Definition

In this case the command being defined is called `\documentclass` and text typed *like this* (e.g. *options* and *class file*) indicates the type of thing you need to substitute. For example, if you want the `article` class file you would substitute *class file* with `article` and if you want the `a4paper` option you would substitute *options* with `a4paper`, like this:

```
\documentclass[a4paper]{article}
```

But more on that later.

2.1 Source Code

The source code is all the text and LaTeX commands that make up an entire document. The source code is typed in using a text editor, and saved with the file extension `.tex`. The source code may be contained in just one file, or it might be split across several files.

2.2 DVI File (or Output File)

The LaTeX application will convert your source code into typeset output which will be written to a device independent (DVI) file. This file can then be viewed using a DVI viewer. `MiKTeX` comes with the DVI viewer `YAP`. If you are using the X Window System (under UNIX or Linux etc), the DVI viewer is called `xdvi`.

2.3 Commands (also called "Macros" or "Control Sequences")

A command usually begins with a backslash, (e.g. `\today`) and is used to tell LaTeX to do a particular thing at that point in the document. For example, `\today` will print the current date, `\twocolumn` will start a new page, and change to a two column format, `\LaTeX` will print the LaTeX logo: LaTeX. Most LaTeX commands have fairly self-explanatory names. (For example, `\rightarrow` prints an arrow pointing to the right, `\chapter` starts a new chapter.) All commands are case-sensitive, so `\gamma` and `\Gamma` have different meanings.

There is one command that you must use in every document you create, and that is the `\documentclass` command. This command must be placed at the very start of your document, and indicates what type of document you are creating. This command takes an argument, and is described in more detail in Chapter 4.

2.4 Grouping

Segments of code may be grouped by placing it within `{` and `}` (curly braces). Most commands that occur within a group will be local to that group. For example, `\bfseries` changes the font weight to bold, so the following segment of code:

↑Input

```
Here is some text. {This text \bfseries is in a
group.} Here is some more text.
```

↓Input

will appear in the typeset document looking like:

Here is some text. This text **is in a group.** Here is some more text. Output

As can be seen, the font change only stays in effect until it reaches the end of the group (signified by the closing curly brace }.)

2.5 Arguments (also called "Parameters")

Some commands take one or more arguments. This allows you to give LaTeX additional information, so that it is able to carry out the command. There are two types of arguments: mandatory and optional.

2.5.1 Mandatory Arguments

Mandatory (or compulsory) arguments are arguments that *have* to be specified. Examples:

1. If you want to start a new chapter, you need to use the `\chapter` command, but you also need to tell LaTeX the title of this new chapter. So the `\chapter` command takes one mandatory argument that specifies the title. For example, the following code:

   ```
   \chapter{Some Definitions}
   ```
   Input

   was used to generate the heading for Chapter 2 of this document.

2. The command `\textbf` typesets its argument in a bold font (as opposed to the declaration `\bfseries` which switches to a bold font.) The following code:

   ```
   \textbf{Some bold text.}
   ```
   Input

   will look like:

   Some bold text. Output

Notes

1. LaTeX takes the first object following the command name as the argument, which is why the argument has to be grouped. Suppose the last example above didn't have a group, so instead the code was:

   ```
   \textbf Some bold text.
   ```
   Input

   then only the 'S' would be the argument because it's the first object following the command, in which case the output would look like:

   Some bold text. Output

2. If you want the argument to be blank, use empty braces: `{}`. For example, suppose you want to have a chapter without a title[1] you would need to do:

```
\chapter{}
```
Input

2.5.2 Optional Arguments

Some commands may have one or more optional arguments. Unlike mandatory arguments, optional arguments must always be enclosed in square brackets `[ ]`. For example, the command `\\` starts a new line. So the following segment of code:

```
Line one\\ Line two.
```
Input

will produce the following output:

↑Output

Line one
Line two.

↓Output

However the `\\` command also has an optional argument that allows you to specify how big the gap between the two lines should be. So the following segment of code:

```
Line one\\[1cm] Line two.
```
Input

will produce the following output:

↑Output

Line one

Line two.

↓Output

Incidentally, note the difference between the previous example, and the following example:
Code:

```
Line one\\{[1cm]} Line two.
```
Input

Output:

↑Output

Line one
[1cm] Line two.

↓Output

[1] The numbers for chapters, sections etc are automatically inserted by LaTeX, so this example would produce a numbered chapter without a title.

In this example the `[1cm]` has been placed inside a group, so it is no longer considered to be an optional argument, and since the command `\\` does not take a mandatory argument, the `[1cm]` is simply interpreted as ordinary text.

Here's another example: The command `\framebox` takes a mandatory argument and an optional argument. `\framebox` puts a frame around the contents of its mandatory argument:
Code:

```
\framebox{Some Text}
```
Input

Output:

Some Text
Output

The optional argument can be used to make the box a specified width:
Code:

```
\framebox[4cm]{Some Text}
```
Input

Output:

Some Text
Output

And there's a second optional argument that specifies the justification of the text (left, right or centred) within the box:
Code:

```
\framebox[4cm][r]{Some Text}
```
Input

Output:

Some Text
Output

In general, if a command has both optional and mandatory arguments, the optional arguments are usually specified first (although there are a few exceptions.)

2.6 Declarations

The term declaration is used to refer to a command that affects the document from that point onwards. The declaration itself does not produce any text, and its effect can be localised by placing the declaration within a group. For example, `\bfseries` is a declaration that switches the current font weight to bold:

```
Here is some normal text.
\bfseries Here is some bold text.
```
↑Input
↓Input

will appear in the typeset document looking like:

Here is some normal text. **Here is some bold text.**
Output

2.7 Environments

An environment is a block of code contained within the commands `\begin{`*env-name*`}` and `\end{`*env-name*`}`, where *env-name* is the name of the environment. The block of code is then formatted in a method specific to that environment. For example, the `bfseries`[2] environment will typeset the contents of the environment in a bold font. The following code:

Input

```
\begin{bfseries}
Here is some bold text.
\end{bfseries}
```

Input

will appear in the typeset document looking like:

Output

Here is some bold text.

Some environments also supply commands that may only be used within that environment. For example, the `itemize` environment provides a command called `\item` so that you can specify individual items within an unordered list.

Example:

Input

```
Shopping List:
\begin{itemize}
\item Cabbages
\item Bananas
\item Apples
\end{itemize}
```

Input

will produce the following output:

Output

Shopping List:

- Cabbages
- Bananas
- Apples

Output

2.8 Preamble

The preamble is the part of the source code that comes between the `\documentclass` command and `\begin{document}` (the start of the `document` environment). Only a few special commands may be placed in the preamble, and there are a few special commands that may only go in the preamble.

[2] note there is no backslash in the environment name

```
\documentclass{...}
```

⟵ This bit in here is the preamble.

```
\begin{document}
```

2.9 Class File

The class file (`.cls`) defines the page layout, heading styles and various commands and environments needed for a particular style of document. The class file is specified using the command

`\documentclass[`*options*`]{`*class-name*`}` Definition

where *class-name* is the name of the file without the `.cls` extension. All LaTeX documents must start with this command.

2.10 TeX

TeX is the typesetting language written by Donald Knuth. Plain TeX is a bit complicated to use, unless you want to write a very basic document, so Leslie Lamport wrote a format of TeX called LaTeX to make it a bit easier to use. You can think of LaTeX as a go-between converting your instructions into TeX. This document mostly uses the term LaTeX, even if the matter is more general to TeX, to avoid complicating matters. Some error messages you may see will be LaTeX messages, some will be TeX messages. LaTeX error messages tend to be a bit easier to understand than TeX messages.

Chapter 3

From Source Code to Typeset Output

Every time you want to create or edit a LaTeX document, there are three basic steps you will always need to follow:

1. Write or edit the source code
2. Pass the source code to the LaTeX application ("LaTeX the document")
   - If there are any error messages, return to Step 1
   - If there are no error messages, a DVI file is created.
3. View the DVI file to check the result. If you need to modify your document, go back to Step 1.

You will therefore need:

1. A text editor or front-end (to perform Step 1), see below.
2. The TeX/LaTeX installation (to perform Step 2). There are a number available, however a popular choice for Windows is `MiKTeX`, which is free and can be downloaded from the TeX Archive [6] in the `systems/win32/miktex` directory and is easy to install. Simply follow the installation instructions. Default values are provided if you are unsure what option to choose, but if you have any difficulties, contact your system adminstrator. Note that even if you are using a front-end, you must first install MiKTeX (or some other TeX/LaTeX installation). If you are using UNIX or Linux, a popular choice is `teTeX`, this can be downloaded from the `systems/unix` or `systems/linux` directories.
3. A DVI viewer (to perform Step 3). The TeX/LaTeX installation should come with a DVI viewer. (MiKTeX comes with `YAP`.) It is also possible to convert your DVI file into PostScript (`.ps`) or Acrobat (`.pdf`) format, in which case you will need `GSView` or `Acrobat`, respectively, to view the files[1]. By converting your output to PostScript or PDF, you can enhance the functionality of LaTeX allowing you to perform operations such as rotating text (See Section 6.1.1 for further details). If you use PDFLaTeX to generate a PDF document, you can also create active links (see *The LaTeX Web Companion* [5]

[1] `GSview` can also display PDF files, but any links in the document will be inactive

for more information, or if you'd rather a brief on-line introduction you can try *Creating a PDF Document using PDFLaTeX*[2]).

Documented below are instructions of how to use LaTeX on Windows using three different methods:

1. Using `notepad` as a text editor, the `MS-DOS Prompt` to access LaTeX, and `YAP` to view the DVI file.
2. Using the front-end `TeXnicCenter` (free) to perform all three steps.
3. Using the front-end `WinEdt` (shareware) to perform all three steps.

Using `notepad` and the `MS-DOS Prompt` is fiddly and prone to human error, however it is useful to know, just in case there is a situation that the front-end can't handle (e.g. you want to LaTeX a file that doesn't have a `.tex` extension, which may happen if you want to install new packages on your system — some front-ends allow you to do this, others may not). I would therefore strongly recommend that you use one of the front-ends (`TeXnicCenter` or `WinEdt`) rather than using `notepad`. `TeXnicCenter` and `WinEdt` are both easy to use, although `TeXnicCenter` has the advantage of being free.

If you are using UNIX or Linux, follow the instructions for using `notepad` and the `MS-DOS` prompt, but sustitute `notepad` for your favourite text editor (e.g. `vim` or `emacs`), use a terminal instead of the `MS-DOS Prompt`, and use `xdvi` instead of `YAP`.

3.1 Notepad, MS-DOS Prompt, YAP

`Notepad` is a very basic text editor that comes with Windows. It is usually found through the Start menu:

Start → Programs → Accessories → Notepad

Once you have opened up `notepad`, you can start to type in your source code. (See Figure 3.1.)

Care needs to be taken when saving the document, as `notepad` automatically tries to add the extension `.txt` to any file you save, whereas all LaTeX files must have the extension `.tex`. You can force `notepad` to do this by placing the filename in double quotes in the "Save as" dialogue box. (See Figure 3.2.)

You can check to see if your file has been saved correctly by looking at the directory viewer. If the file has been saved correctly, it should look something like Figure 3.3.

If the file has the incorrect `.txt` extension added to it, it will probably look something like Figure 3.4. In this case the icon indicates that this is an ordinary text file. If you look at the file's properties you will see that its name is actually `sample1.tex.txt`, which is incorrect.

Step 1 is now complete. Time to move on to Step 2: passing the source code to LaTeX. To do this you will need to run the `MS-DOS Prompt`. This is usually found in:

Start → Programs → MS-DOS Prompt

or

Start → Programs → Accessories → MS-DOS Prompt

The command prompt should look something like Figure 3.5.

The first thing you need to do is to change to the directory where you saved your file (otherwise LaTeX won't know where the file is.) You can do this using the

[2] http://theoval.cmp.uea.ac.uk/~nlct/latex/pdfdoc/

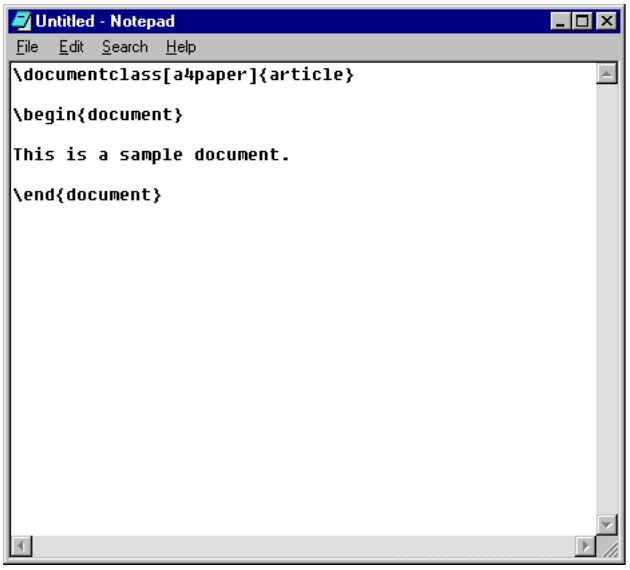


Figure 3.1: Using notepad

`cd` command. I saved my file in the directory `My Documents\Nicky\samples` on the `C` drive. Since the directory name has a space in it, it will need to be enclosed in double quotes. At the command prompt, I would then have to type

```
cd "c:\My Documents\Nicky\samples"
```

as shown in Figure 3.6.

You can now pass the source code to LaTeX. I called my file `sample1.tex` so I would need to type

```
latex sample1.tex
```

at the command prompt, as shown in Figure 3.7. (You don't have to specify the `.tex` extension, as LaTeX will automatically assume your file has the correct extension).

If there are no errors in the document, you should see something like the output shown in Figure 3.8. (If you do get an error message, check the list of common errors in Chapter 13.)

The second to last line

```
Output written on sample1.dvi (1 page, 248 bytes).
```

indicates that the resulting typeset document has been saved as the file `sample1.dvi` and it is one page long. Numbers appearing in square brackets, e.g. `[1]`, indicate which page LaTeX is currently processing. In this case, there is only one page. The last line to appear on screen indicates that information about this LaTeX run has been written to the log file `sample1.log`, which you can look at using `notepad`. You can see these new files by having a look at the directory viewer, as shown in Figure 3.9.

You can view the typeset output by loading the file `sample1.dvi` into `YAP`. You can do this either by double clicking on its icon, or by typing

```
yap sample1.dvi
```

at the command prompt. (See Figure 3.10.)

You will then see the final output, as shown in Figure 3.11.

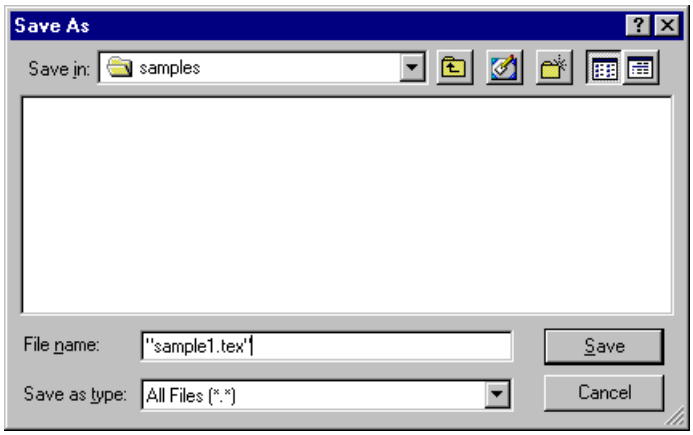


Figure 3.2: Saving your document

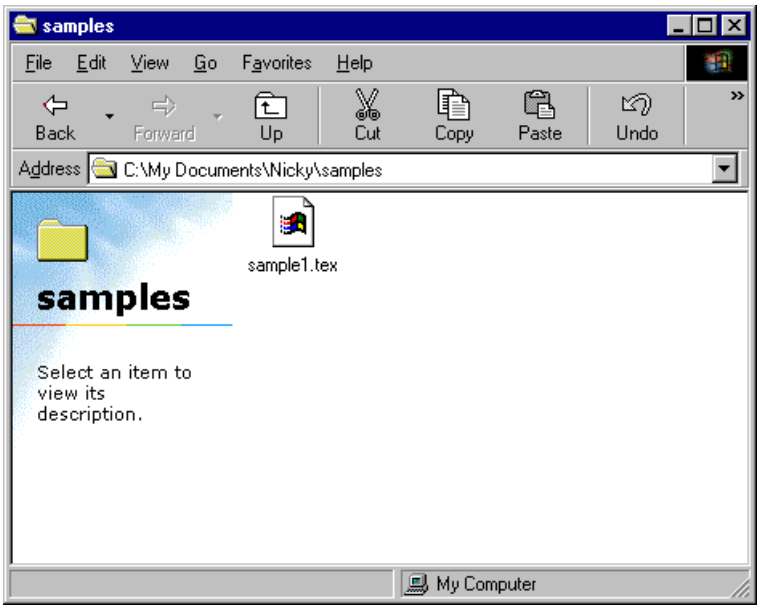


Figure 3.3: File saved correctly

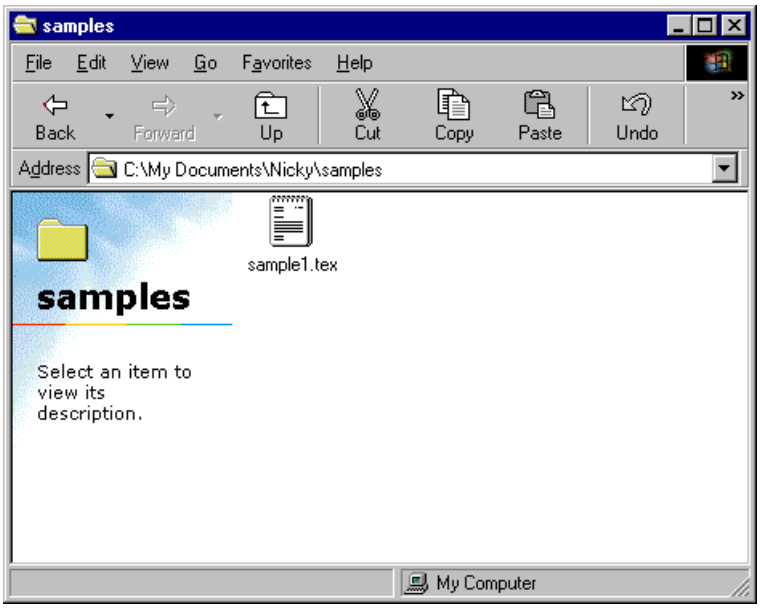


Figure 3.4: File saved incorrectly

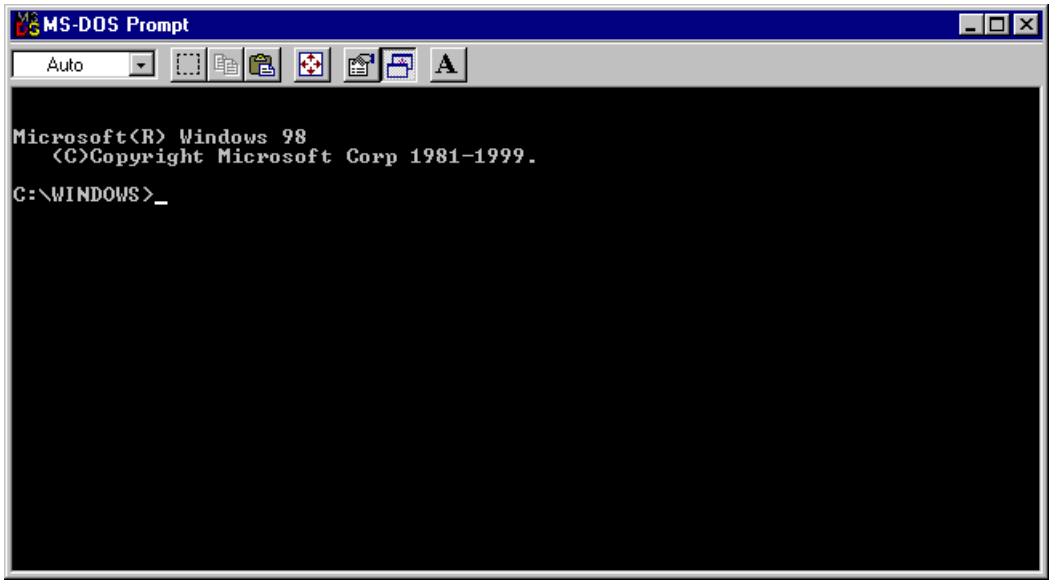


Figure 3.5: MS-DOS Prompt

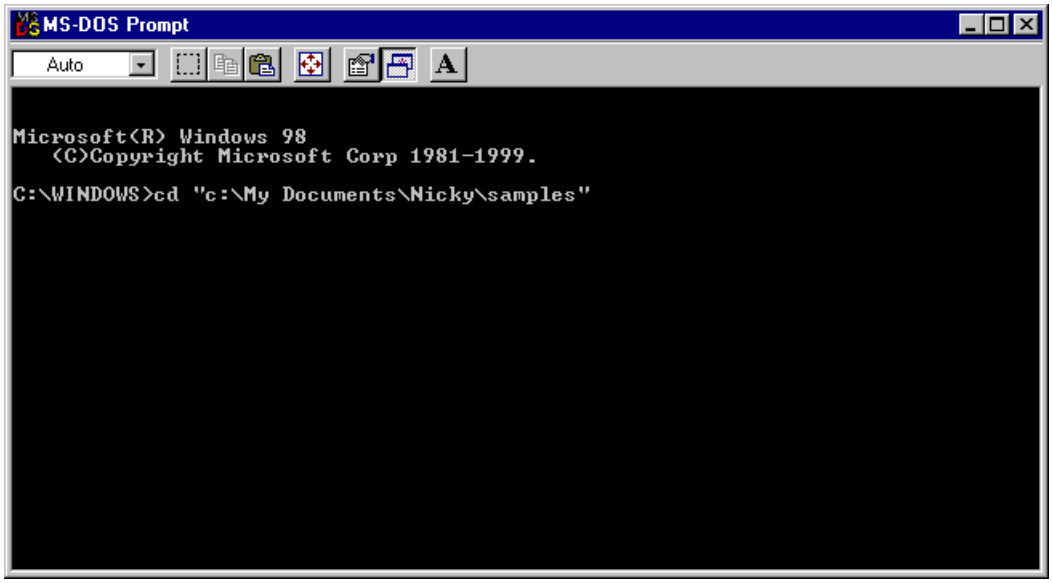


Figure 3.6: Changing Directory

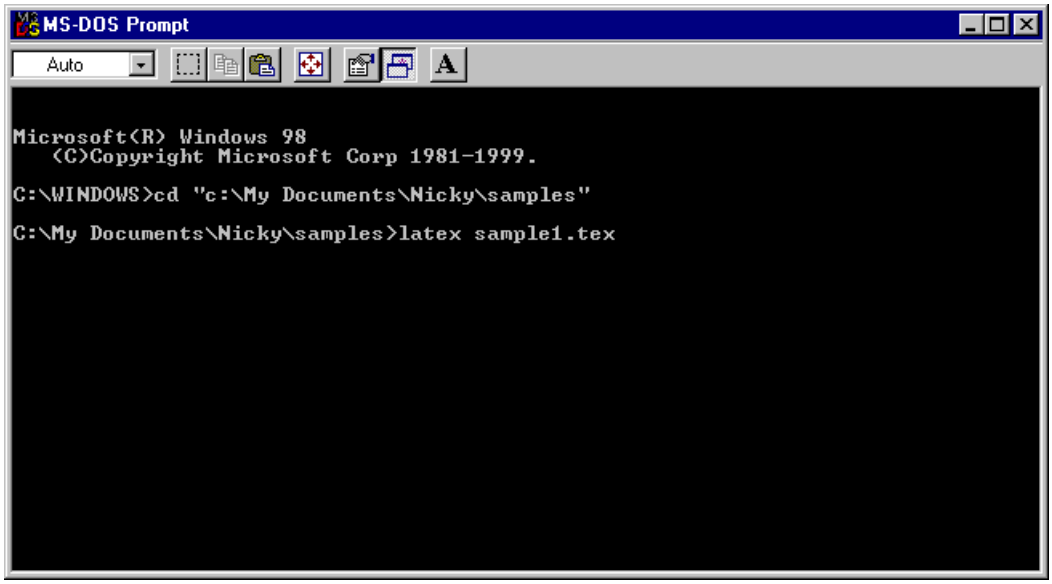


Figure 3.7: Using LaTeX

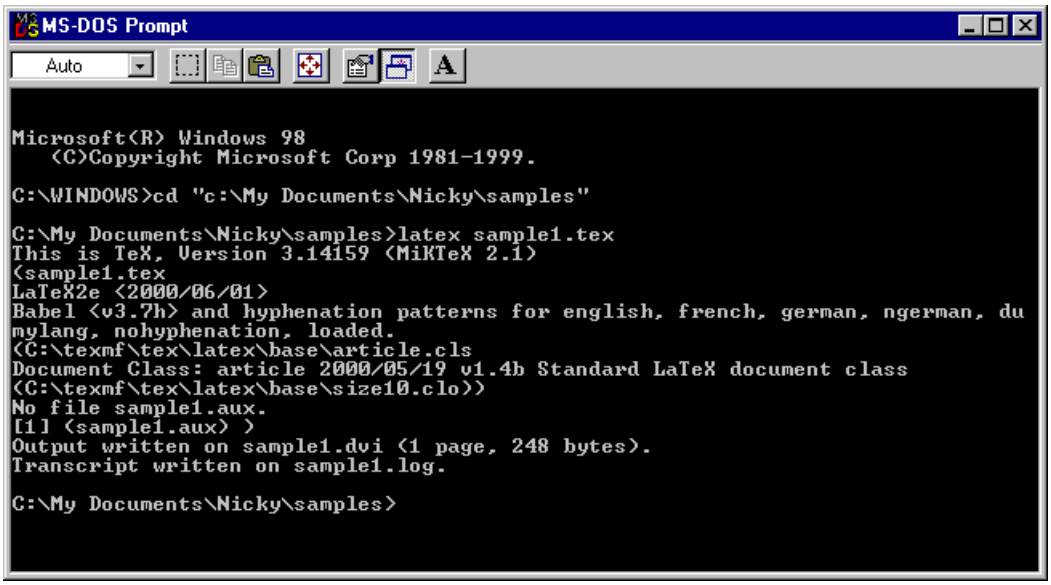


Figure 3.8: LaTeX output

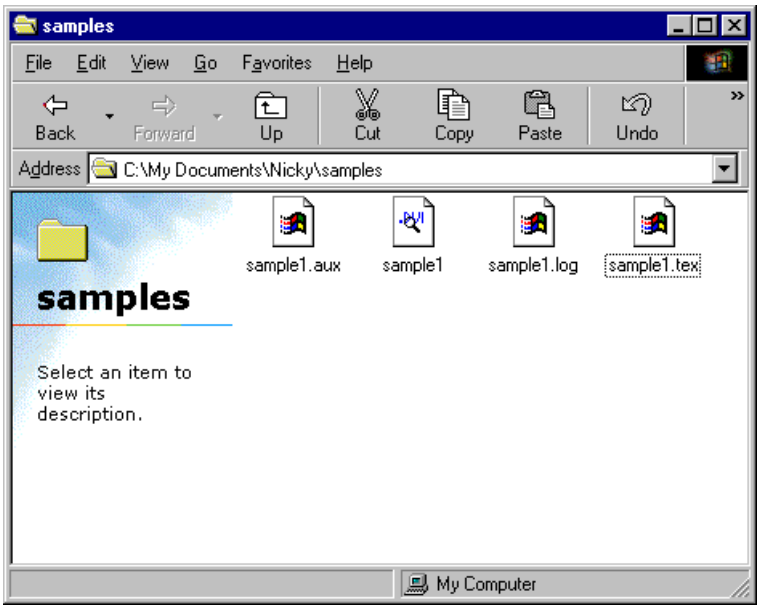


Figure 3.9: Three New Files Created by LaTeX

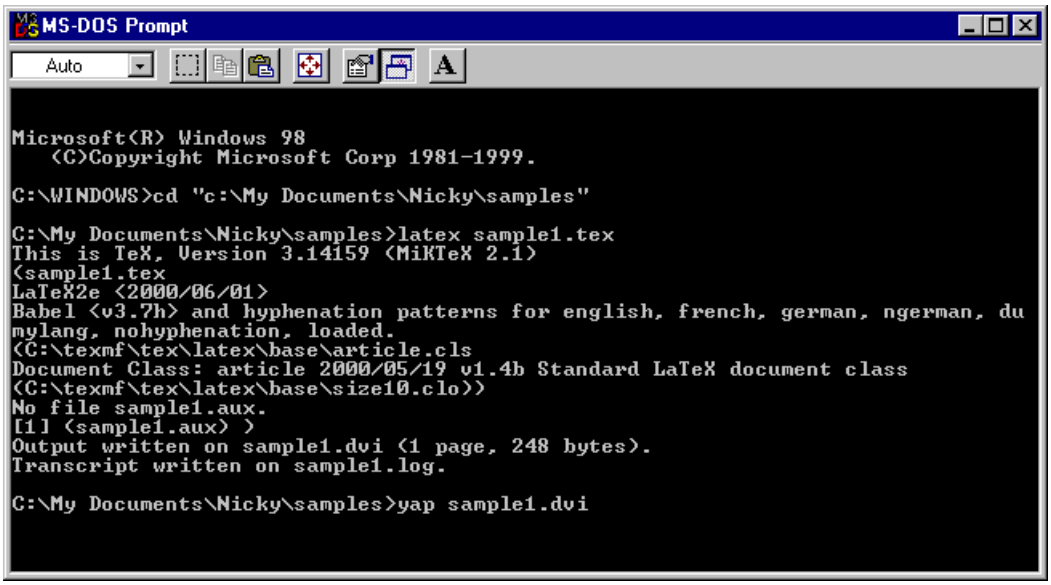


Figure 3.10: Loading output file into YAP

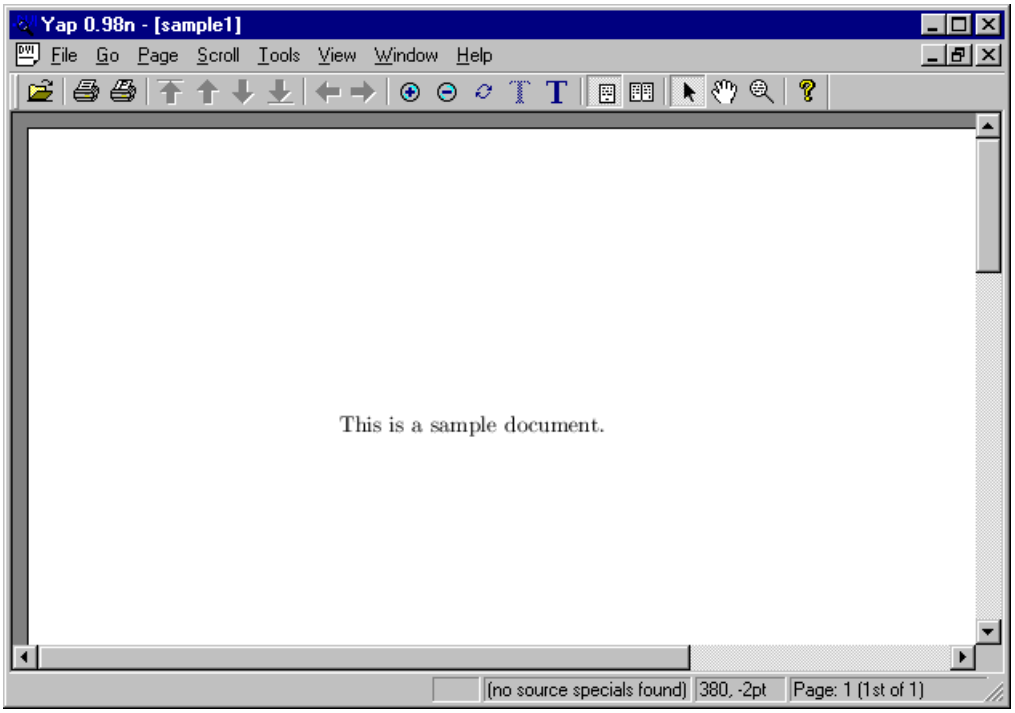


Figure 3.11: Viewing typeset document

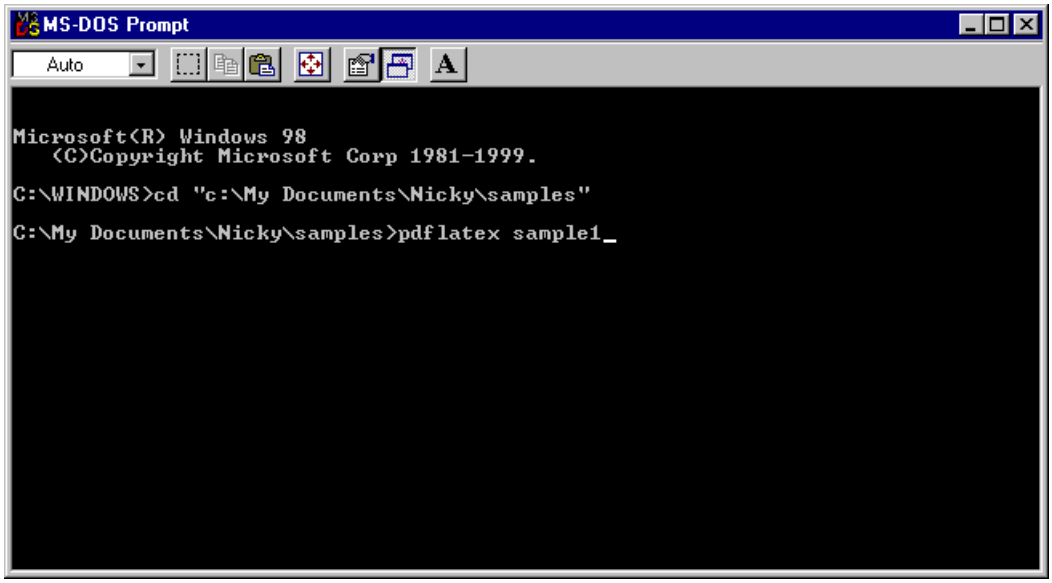


Figure 3.12: Using PDFLaTeX.

If you like, you can then convert your DVI file into a PostScript file by using `dvips`. To do this, type the following in the `MS-DOS Prompt` window:

```
dvips -o sample1.ps sample1.dvi
```

You can then view the PostScript file using `GSView`[3] which can be downloaded from the TeX Archive [6].

Alternatively, you can create a PDF document instead by using PDFLaTeX instead of LaTeX:

```
pdflatex sample1.tex
```

as shown in Figure 3.12. (Again the extension may be omitted.)

The output is shown in Figure 3.13.

The new PDF file `sample1.pdf` can now be loaded into `Acrobat`[4], as shown in Figure 3.14.

Each time you want to edit the document, you will have to go back to Step 1 (although you shouldn't need to worry about changing directory anymore, unless you exit the `MS-DOS Prompt`.) This method can be rather cumbersome, however life is made a lot easier by using a front-end, such as `WinEdt` or `TeXnicCenter`.

3.2 TeXnicCenter

`TeXnicCenter` is an application that enables you to edit LaTeX source code, and simply click on a button to pass the source code to LaTeX, and then click on another button to view the resulting typeset document. This alieviates the problems encountered using `notepad` and the `MS-DOS Prompt` detailed in Section 3.1.

`TeXnicCenter` is free and can be downloaded from the TeX Archive [6] in the `systems/win32/TeXnicCenter/` directory. Note that you must have a TeX/LaTeX distribution installed before you install `TeXnicCenter`. Once you have downloaded

[3] `ghostview` or `ggv` if you are using UNIX or Linux

[4] You can either use `xpdf` or `acroread` if you are using UNIX or Linux

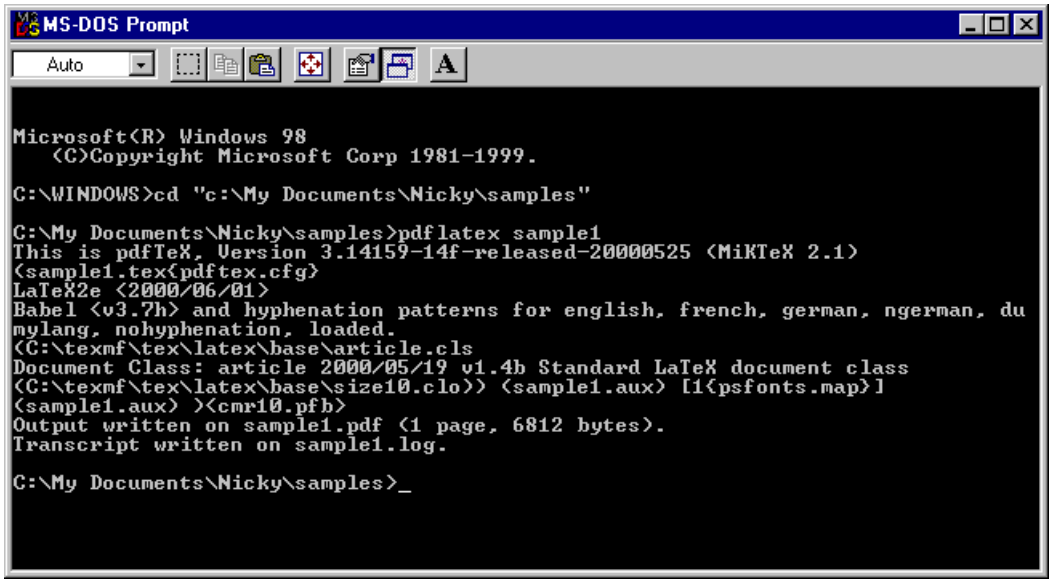


Figure 3.13: PDFLaTeX Output.

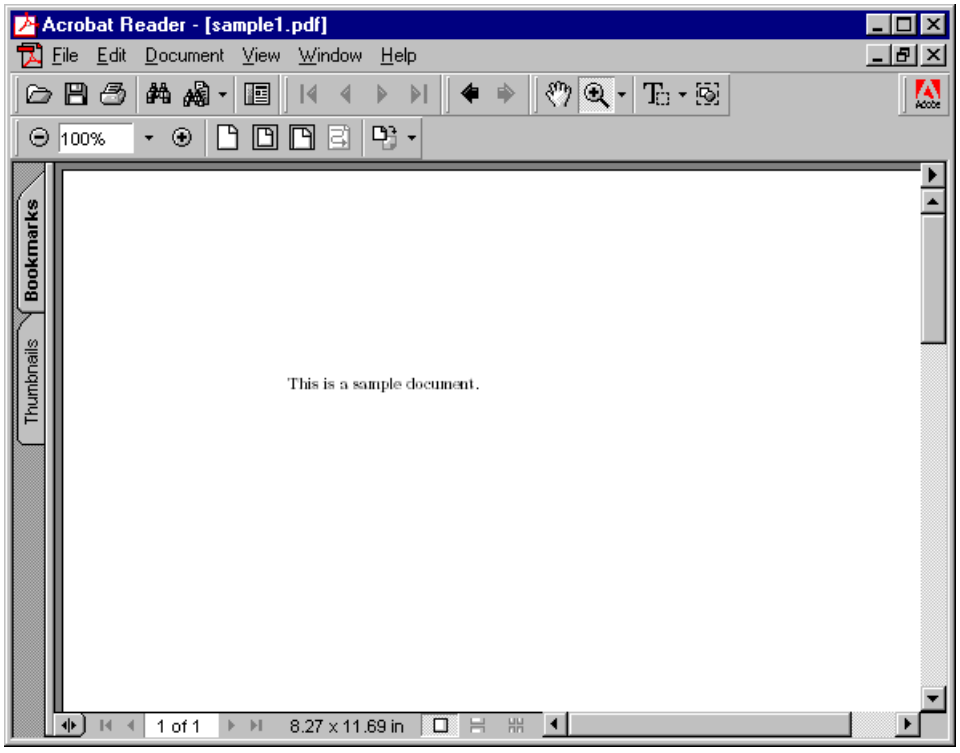


Figure 3.14: Viewing PDF file in Acrobat

the `TeXnicCenter` setup file[5], run it by double clicking on its icon in the directory viewer. I recommend that you use the default settings. If you have any problems installing `TeXnicCenter`, contact your systems administrator.

Once the installation is complete, you can then run `TeXnicCenter` from the Start Menu:

Start → Programs → TeXnicCenter → TeXnicCenter

Firstly you should see the tip of the day window (Figure 3.15.)

Figure 3.15: TeXnicCenter Tip of the Day Window

You can close this window, and then, if this is the first time you are using `TeXnicCenter` you will have to use the configuration wizard to set up `TeXnicCenter` correctly. I would recommend that you choose the default settings. (Select Next, Next and then Finish.)

Now you are ready to use `TeXnicCenter`. It should look like Figure 3.19.

To start a new project select File → New Project. This will open the window shown in Figure 3.20.

Enter a name for your project, and specify the directory where you want to save your work. For example, I shall call my project "example" and I want to save it in `c:\My Documents\Nicky\example` (see Figure 3.21.)

Select the "Empty Project" icon, and click on "Okay". You should now see something like Figure 3.22.

You can now start typing the source code (we'll cover this later). See Figure 3.23.

Save it by either clicking on the save icon  or select File → Save

Now select what type of output you want (DVI, PDF or PostScript) see Figure 3.24. If this box is blank, then it's possible that you didn't complete all the steps in the configuration wizard described above.

Now click on the build output icon  or select Build → Build Output. The transcript will be written in the window at the bottom (see Figure 3.25) This tran-

[5] currently called `TXCSetup_1Beta6_21.exe`

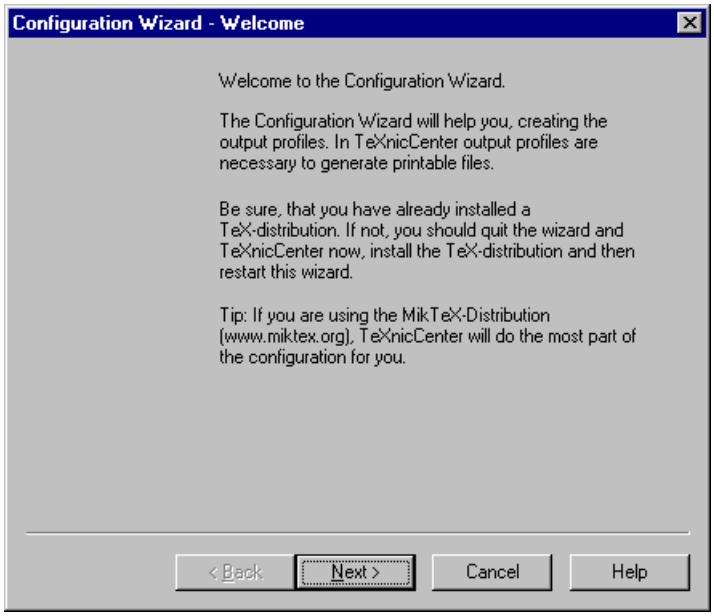


Figure 3.16: TeXnicCenter Configuration Wizard

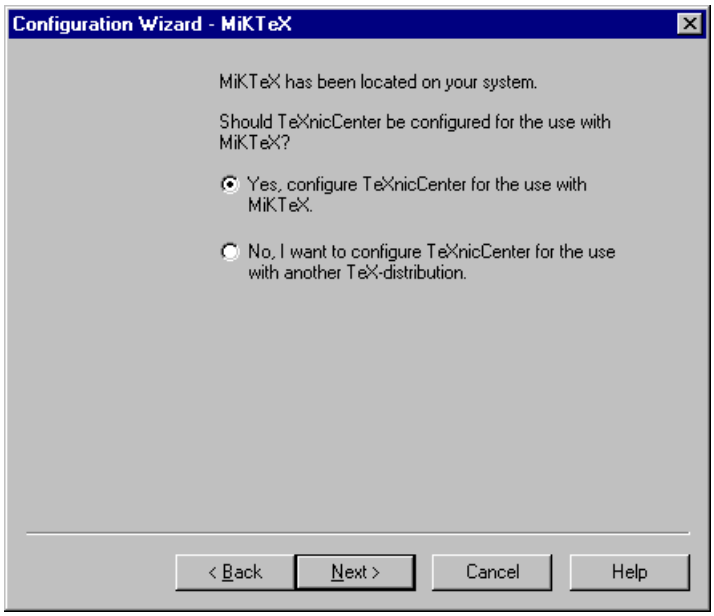


Figure 3.17: TeXnicCenter Configuration Wizard

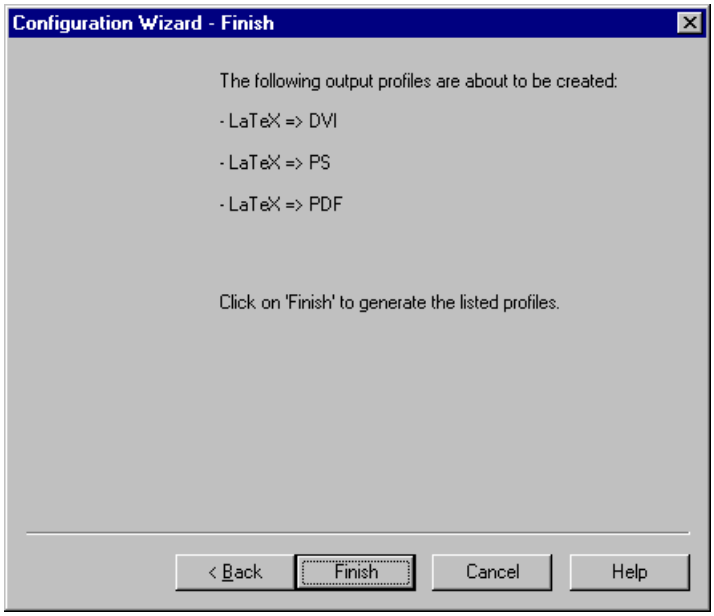


Figure 3.18: TeXnicCenter Configuration Wizard

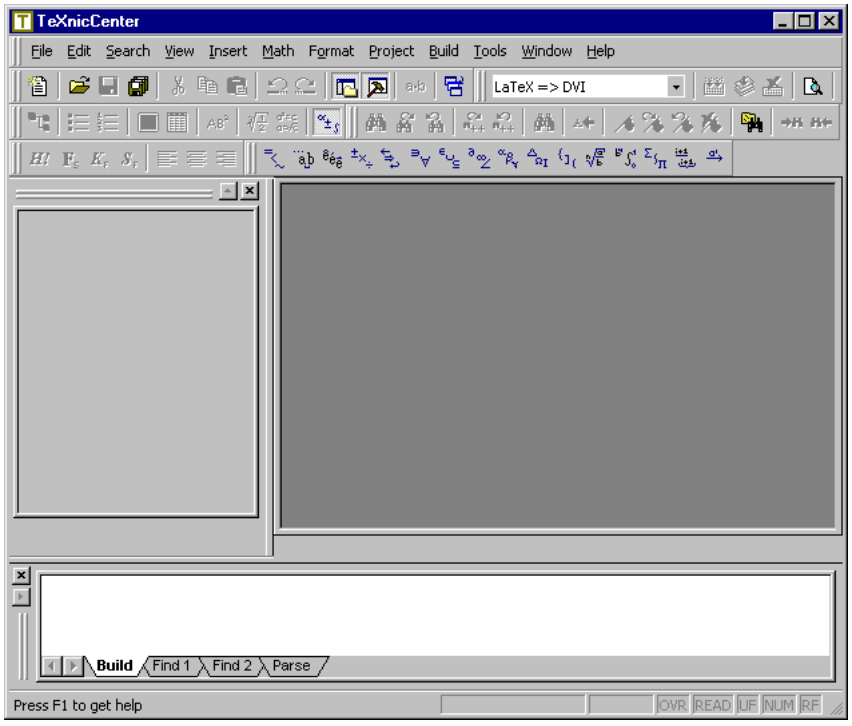


Figure 3.19: TeXnicCenter

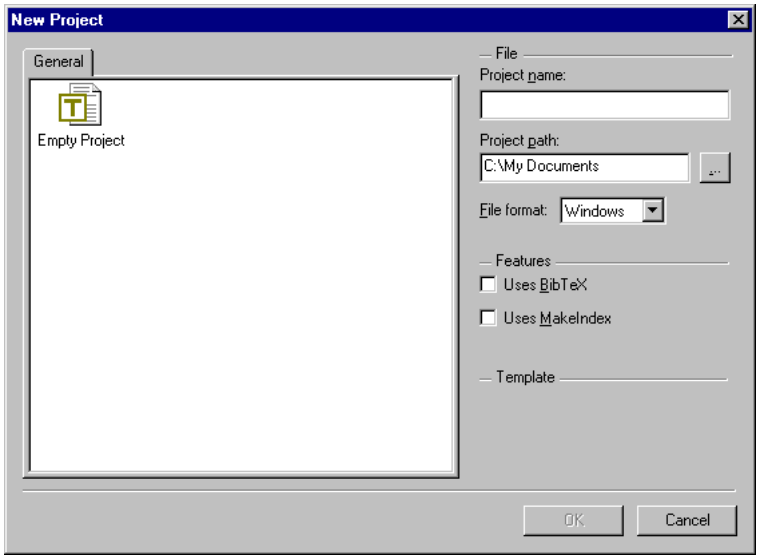


Figure 3.20: New Project Dialog Box

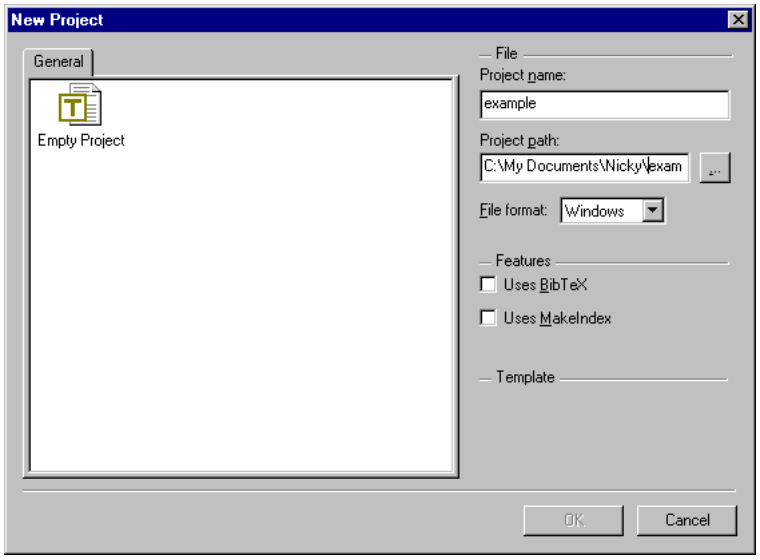


Figure 3.21: New Project Dialog Box

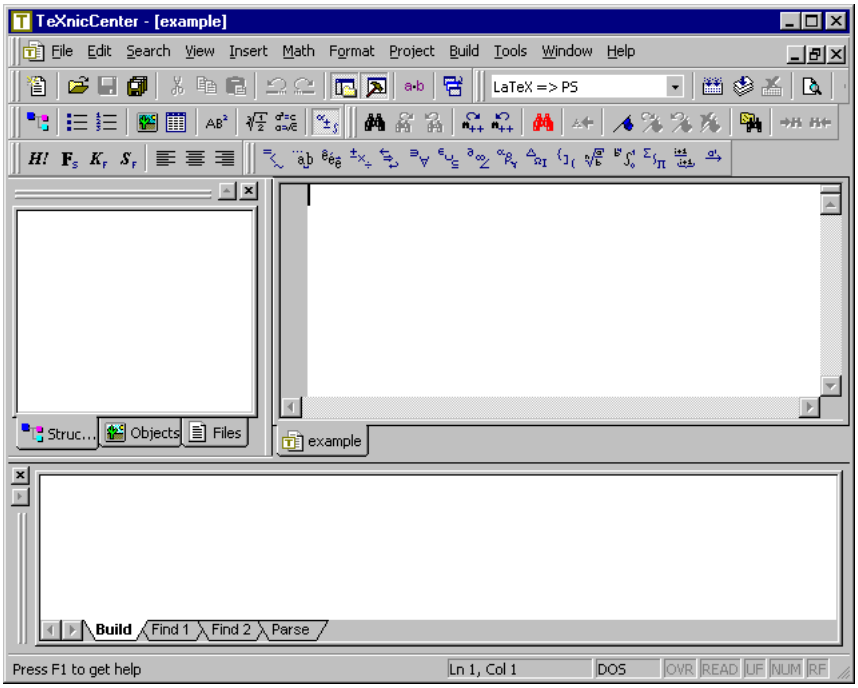


Figure 3.22: TeXnicCenter — New Project Started

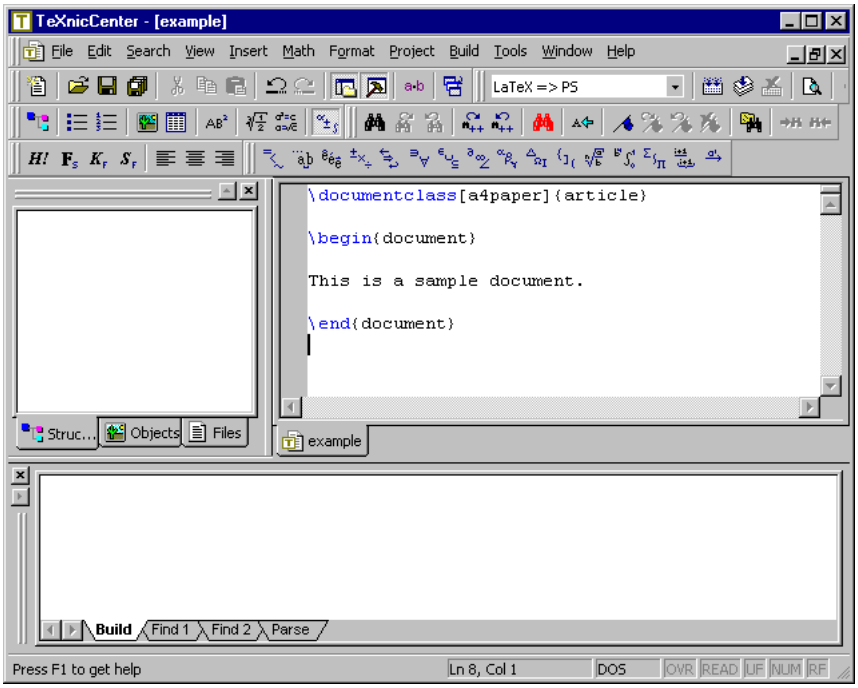


Figure 3.23: TeXnicCenter — Typing in Source Code

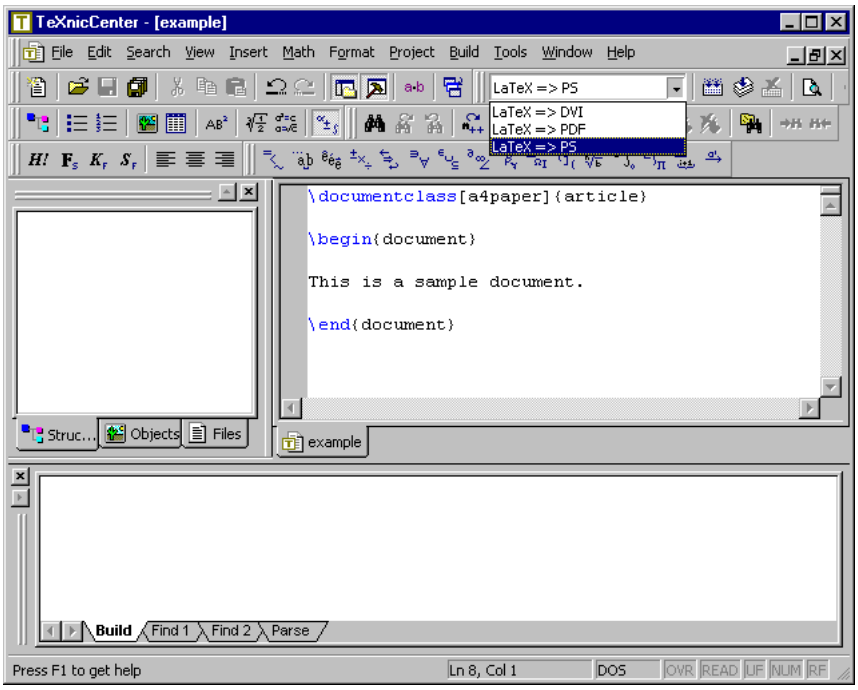


Figure 3.24: TeXnicCenter — Selecting Output Type

script should be the same as described on page 13 onwards. If you have selected `LaTeX => PDF`, then `TeXnicCenter` will use PDFLaTeX instead of LaTeX. If you have selected `LaTeX => PS`, then `TeXnicCenter` will use LaTeX followed by `dvips` (as in Figure 3.25). The `dvips` messages will follow on from the LaTeX messages. (If you selected the BibTeX or MakeIndex features when you initialised the project, Figure 3.21, then `TeXnicCenter` will also use the BibTeX and `MakeIndex` applications.)

To view the document, click the View Output button  . (Note that if you have selected `LaTeX => PDF` or `LaTeX => PS` you will need Adobe Acrobat or GSView, respectively, to view the output file.)

If there are any errors, you can select Build → Next Error and it will show you where the error has occured (See Figure 3.26). If you do have any errors, check Chapter 13.

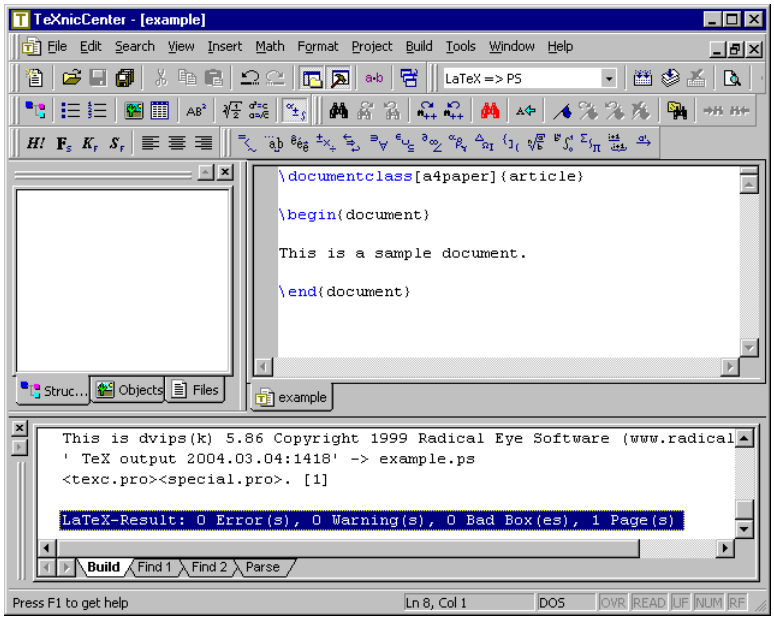


Figure 3.25: TeXnicCenter (using LaTeX and `dvips`)

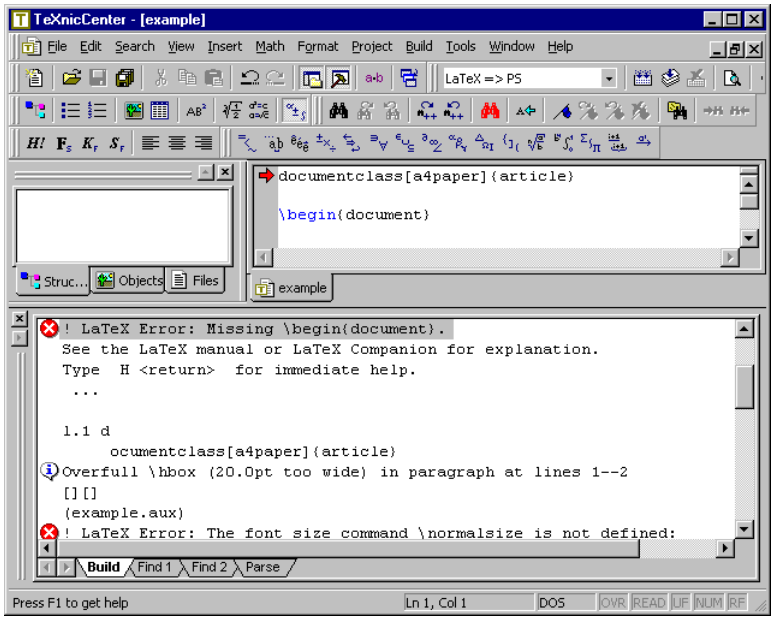


Figure 3.26: TeXnicCenter — Showing Error

3.3 WinEdt

`WinEdt` (not to be confused with `WinEdit` which is a completely different application) is an application that enables you to edit LaTeX source code, and simply click on a button to pass the source code to LaTeX, and then click on another button to view the resulting typeset document. This alieviates the problems encountered using `notepad` and the `MS-DOS Prompt` detailed in Section 3.1.

`WinEdt` is shareware: it can be downloaded from the TeX Archive [6] in the `systems/win32/winedt` directory and evaluated for a trial period of 31 days, after which, if you want to continue to use it, you must pay the registration fee. Details of prices and types of licence available can be found at `http://www.winedt.com/`. Again, you must have a TeX/LaTeX distribution installed before you start. `WinEdt` is fairly easy to install. First unpack all the files, and then run the `setup.exe` application. I recommend that you use the default settings. If you have any problems installing `WinEdt`, contact your system administrator.

To run `WinEdt`, select `WinEdt` from the start menu:

Start → Programs → WinEdt → WinEdt

It should look like Figure 3.27

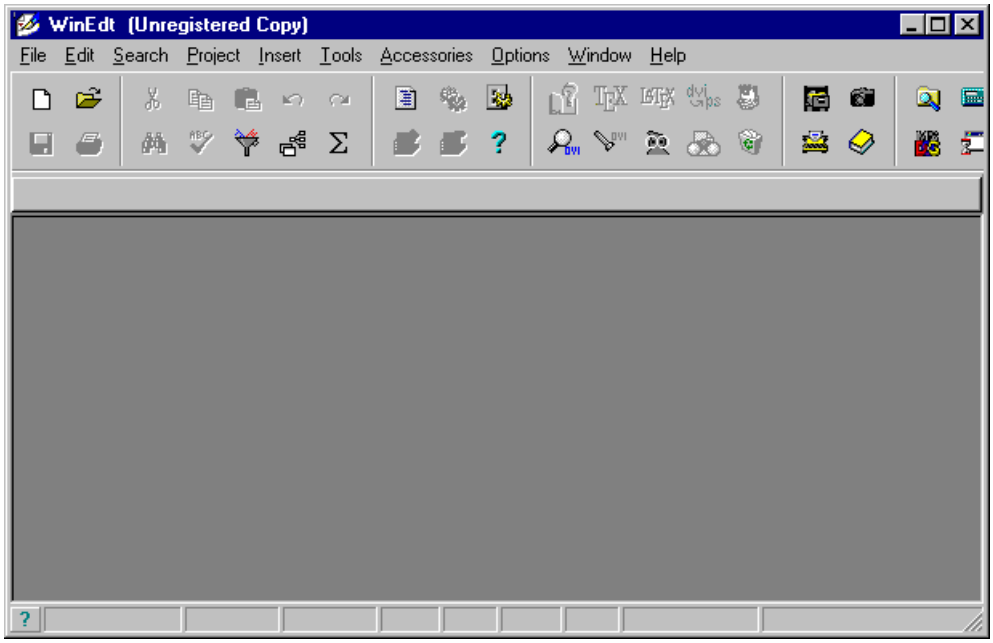


Figure 3.27: WinEdt

Click on the 'New Document' button or select File → New. You can now start typing your source code into the `WinEdt` window, as shown in Figure 3.28

You can now save your document using the File → Save as menu. Select the file type to be `TeX`, and type in the name of your file, e.g. `sample1.tex`. See Figure 3.29.

To LaTeX your document, simply click on the LaTeX button  . The output will appear in an MSDOS Prompt window (see Figure 3.30).

To view your typeset document, click on the "view DVI" button  .

You can convert your DVI file to PostScript by clicking the  button. If

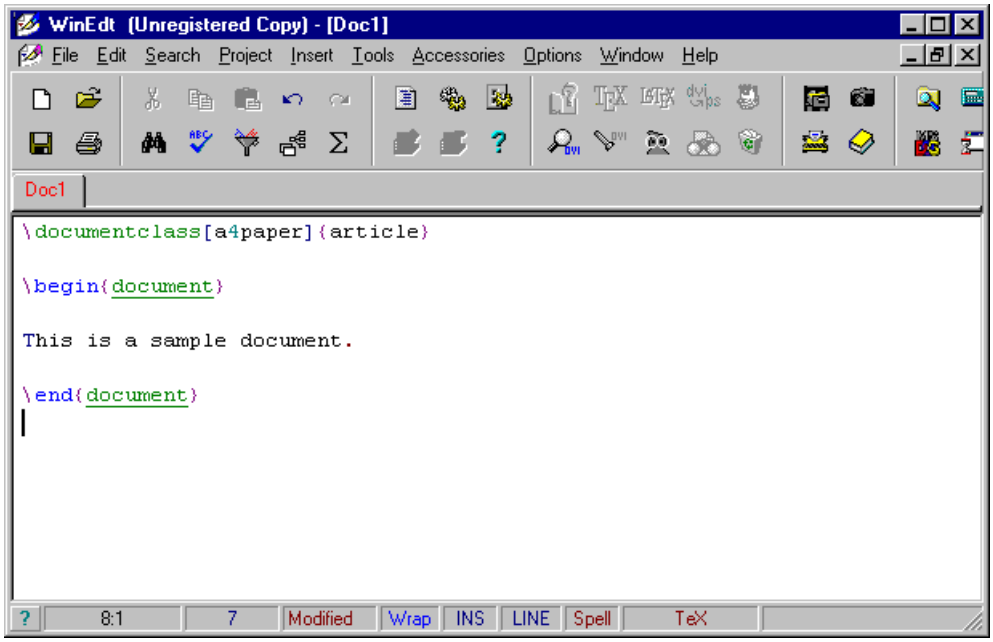


Figure 3.28: WinEdt

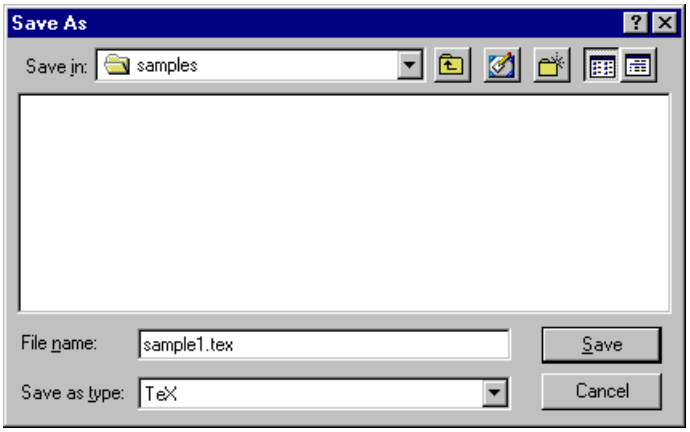


Figure 3.29: WinEdt — Saving the File

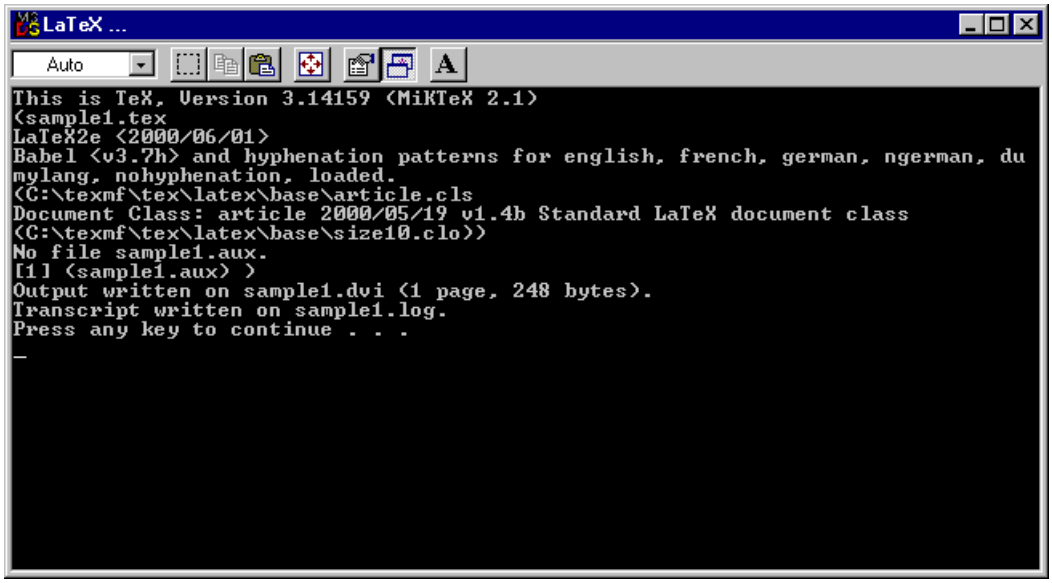


Figure 3.30: WinEdt — LaTeX Output

you have `GSView` installed, you can then view the PostScript file by clicking on the  button.

Depending on which version of WinEdt you have installed, there may also be a PDFLaTeX button which you can click on to create a Portable Document Format (`.pdf`) document. If not, you can click on the  button to open up an MS-DOS Prompt window[6], and you can use the commands listed on page 19.

[6] `WinEdt` should automatically set the correct directory, so you shouldn't need to worry about changing directory

Chapter 4

Creating a Simple Document

Let's now look at how to actually write the source code. The very first line of any document that you create must have the command:

`\documentclass[`*option-list*`]{`*class-name*`}` Definition

This tells LaTeX what type of document you want to create (e.g. an article, a technical report, correspondence). The `\documentclass` command takes one mandatory argument *class-name* that specifies the class file. There are a great many available, but the basic ones are: `article` (short documents without chapters), `report` (longer technical documents containing chapters), `book` (for writing books), `letter` (for writing correspondence) and `slides` (for creating slides for use with OHP or data projectors).

We'll be starting with a very simple document, so let's use the `article` class file. In this case the very first line of the source code should be:

```
\documentclass{article}
```

The `\documentclass` command also takes an optional argument *option-list* which should be a comma separated list of options to be passed to the class file. This allows you to override the class file defaults. For example, the `article` class file by default uses US letter paper, but in the UK we would want to use A4. This can be achieved using the option `a4paper`. So you would need to edit the above line to:

```
\documentclass[a4paper]{article}
```

Let's change another option. The normal font size is `10pt` by default, but we have the option to change it to `11pt` or `12pt`, so let's change it to 11pt:

```
\documentclass[a4paper,11pt]{article}
```

You can also change your document so that it is in a two column format using the `twocolumn` option:

```
\documentclass[a4paper,11pt,twocolumn]{article}
```

Note that there must not be any spaces between the options.

After deciding what type of document we want, we now need to specify the contents of the document. We do this in a `document` environment. The document is started with the command:

```
\begin{document}
```

and ended with

```
\end{document}
```

So our source code now looks like:

↑Code

```
\documentclass[a4paper,11pt]{article}

\begin{document}

\end{document}
```

↓Code

Every document you create must have this form. You can't simply start typing the contents of the document. You must firstly specify your class file, and then place the contents of the document inside the `document` environment. It is a common mistake when first starting out to miss out one or more of these three lines.

So far so good, but at the moment we have an empty document, so we won't get any output. Let's now put some text into our document:

↑Code

```
\documentclass[a4paper,11pt]{article}

\begin{document}

This is a simple document.
Here is the first paragraph.

Here is the second paragraph. As you
can       see it's
a very
short document.

\end{document}
```

↓Code

Exercise 1 (Simple Document)

Try typing the above code into your editor (see Chapter 3 if you can't remember what to do.) You can also download a copy of this file, but I would recommend that you try typing it in to give yourself some practice. If you are using `TeXnicCenter`, start a new project as detailed on page 21. Call your project, say, `sample1`.

Things to note while you are typing: Firstly, when you press the return character at the end of the line this end of line character is converted into a space in the output file. So the fact that I have some very ragged lines in my source code has no effect on the final result.

Secondly, multiple spaces are converted into a single space, so the large gap between the words `can` and `see` is no different from having a single space.

Thirdly, a completely blank line will be converted into a paragraph break, but that doesn't mean that you'll have a blank line between your paragraphs in the output. In fact, by default you won't with most class files, although you can override this.

Fourthly, you don't need to worry about the indentation at the start of new paragraphs as this is done automatically (again it is possible to override paragraph indentation, or change the indentation length.)

Once you have typed up your source code, save your file as, say, `sample1.tex` (or just click on the save icon if you are using `TeXnicCenter`) and then pass it to LaTeX (either by typing `latex sample1.tex` in the MS-DOS Prompt, or by clicking on the LaTeX icon in `WinEdt`, or by clicking on the build icon in `TeXnicCenter` as detailed in Chapter 3.) If all goes well, you should see something that looks like the following displayed on the screen:

```
This is TeX, Version 3.14159 (MikTeX 2.1)
(sample1.tex
LaTeX2e <2000/06/01>
Babel <v3.7h> and hyphenation patterns for american, french, german,
ngerman, italian, nohyphenation, loaded.
(C:\texmf\tex\latex\base\article.cls
Document Class: article 2000/05/19 v1.4b Standard LaTeX document class
(C:\texmf\tex\latex\base\size11.clo))
No file sample1.aux.
[1] (sample1.aux) )
Output written on sample1.dvi (1 page, 376 bytes).
Transcript written on sample1.log.
```

This indicates that your source code has successfully been converted into the typeset output contained in the new file `sample1.dvi`. You can now view this document either by typing `yap sample1.dvi` in the MS-DOS Prompt, or by clicking on the view output button in TeXnicCenter or the view DVI button in WinEdt.

If you have made a mistake in the source code, an error message will be displayed on screen, and the question mark prompt will appear. At this point you can either type `h` for a help message, or type `x` to exit LaTeX and go back to your source code and fix the problem[1]. If you do have an error, consult the list of common mistakes in Chapter 13 for guidance.

4.1 Using Simple Commands

Now let's try adding a few simple commands to our document. The command `\LaTeX` produces the logo LaTeX and the command `\today` prints the current date. LaTeX always ignores any spaces that follow a command name, as it uses the space to indicate the end of the command name. This means that if we want a space to occur immediately after the command, we would need to explicitly say so using the command `\␣` where ␣ indicates a space character. Let's also try using a command that takes an argument. The command

`\footnote{`*text*`}` Definition

takes one argument that specifies the text that should appear in the footnote. This command should be placed where you want the footnote marker to appear.

Exercise 2 (Using Simple Commands)

Try editing the document you created in Exercise 1, so that it looks like the following: (You can download it if you like, but again it is better if you try typing it in yourself)

[1] TeXnicCenter is non-interactive, it will carry on going until it gets to the end. Once it has finished you can locate each error as described on page 26.

↑Code

```
\documentclass[a4paper,11pt]{article}

\begin{document}

This is a simple \LaTeX\ document.
Here is the first paragraph.

Here is the second paragraph. As you
can see it's
a very
short document\footnote{with a footnote}.
This document was created on: \today.

\end{document}
```

↓Code

Now LaTeX your document and view the result. (Remember to check the list of common errors in Chapter 13 if you have a problem.) You should see the LaTeX logo, the footnote marker and the current date. If you scroll down to the bottom of the page, you should see the footnote.

4.2 Special Characters and Symbols

You can use any of the standard characters that you find on your keyboard, except the following 10 symbols:

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

These symbols may only occur in LaTeX commands. We have already used the curly braces `{` and `}`. The percent symbol `%` is a comment character. Everything from the percent symbol up to the end of line is ignored by LaTeX. This means you can have comments in your source code to remind you what a particular part of your code is doing. You have also used the backslash symbol `\` which indicates that you are using a LaTeX command, as in `\LaTeX` or `\today`. The meaning of the other special characters will be covered later.

So what do you do if you want one of these symbols to actually appear in your document? Table 4.1 lists commands that produce these and other symbols. (The symbol ‘ is the backtick symbol, as opposed to the apostrope symbol ’. The backtick symbol usually looks like ` on a keyboard, and on most UK keyboards it is situated to the left of the `1` key. The opening double quote is created using two adjacent backtick symbols, and the closing double quote is created using two adjacent apostrope symbols, this gives 66 and 99 style quotes, which you wouldn't get using the double quote character.)

Ligatures and special symbols are shown in Table 4.2. (Note that, as mentioned in the Introduction, the f-ligatures are automatically converted.) When using a command in the middle of a word, take care that the command doesn't run into the rest of the word. For example, the British spelling of the word “manœuvre” has an œ-ligature in the middle of it. There are several ways to code this in LaTeX:

Table 4.1: Symbols

\textbackslash	\	_	_	\P	¶	-	-
\textasciicircum	^	\$	$	\S	§	--	–
\textasciitilde	~	\{	{	\ldots	...	---	—
\pounds	£	\}	}	\dag	†	?‘	¿
\textregistered	®	\#	#	\ddag	‡	!‘	¡
\texttrademark	TM	\%	%	’	’	’’	”
\copyright	©	\&	&	‘	‘	‘‘	“
\yen	¥	\i	ı	\j	ȷ		

1. Group the command:

```
man{\oe}uvre
```
Input

2. Place a space after the command:

```
man\oe uvre
```
Input

3. Place an empty brace after the command:

```
man\oe{}uvre
```
Input

Each of these three methods produce the same result, but I personally prefer the first method. It is important to make your source code as easy to read as possible, as you may need to edit your document; the first of the above three examples retains the look of a complete word, whereas the second example fragments the word, so although the word is whole in the output, it doesn't read right when you're editing your code. The third example, like the first example, maintains the word's cohesion, but it gives the incorrect impression that the command `\oe` has an argument. However, as I mentioned, this is my personal preference, you should use whichever method you feel most comfortable with, just as long as you don't do the following:

```
man\oeuvre
```

This is incorrect, as LaTeX will interpret it as the command `\oeuvre` which doesn't exist.

Accented letters are created by specifying which accent you want, and what letter to put the accent on. The accent commands are listed in Table 4.3, and each command takes one mandatory argument. The command indicates what accent to use, the argument indicates what letter to put the accent on. You may have noticed in Table 4.1 the commands `\i` and `\j` which produce a dotless i and j (ı and ȷ). You should use these instead of `i` and `j` as the argument to an accent command, since i and j should loose their dot when they have an accent over them. Example:

Table 4.2: Ligatures and Special Symbols

AE	Æ	ae	æ	OE	Œ	oe	œ
fi	fi	ffi	ffi	fl	fl	ffl	ffl
AA	Å	aa	å	L	Ł	l	ł
O	Ø	o	ø	SS	SS	ss	ß

Table 4.3: Accent Commands

Example			Example		
Definition	Input	Output	Definition	Input	Output
\'{*object*}	\'{c}	ć	\={*object*}	\={c}	c̄
\`{*object*}	\`{c}	ò	\.{*object*}	\.{c}	ċ
\^{*object*}	\^{c}	ĉ	\~{*object*}	\~{c}	c̃
\"{*object*}	\"{c}	c̈	\v{*object*}	\v{c}	č
\u{*object*}	\u{c}	c̆	\H{*object*}	\H{c}	c̋
\t{*object*}	\t{cc}	c͡c	\c{*object*}	\c{c}	ç
\d{*object*}	\d{c}	c̣	\b{*object*}	\b{c}	c̲

```
It's na\"{\i}ve to think that eating mouldy p\^{a}t\'{e}
won't result in food poisoning.
```

↑Input

↓Input

It's naïve to think that eating mouldy pâté won't result in food poisoning.

↑Output

↓Output

Exercise 3 (Using Special Characters)

Start a new file (or project if using TeXnicCenter), and see if you can write the source code to create the following output:

↑Output

Item #1: Our travel expenditure came to $2000.00 & our equipment expenditure came to £100.00 plus VAT @ 17.5%.

↓Output

You can download or view the source code if you can't work out how to do it, and remember to check the list of common errors in Chapter 13 if you have a problem.

4.3 Lists

Now you've had a go at using some commands, let's use some environments. A good example of environments are the list making environments. There are three basic list making environments: `itemize` (for unordered lists), `enumerate` (for ordered lists) and `description` (for lists where you want to specify your own label.)

In each of these environments, there is a command

`\item[`*label*`]` Definition

which you need to use to specify each item of the list.

4.3.1 Unordered Lists

Unordered lists are created using the `itemize` environment. For example, the following code:

↑Input

```
\begin{itemize}
\item Animal
\item Vegetable
\item Mineral
\end{itemize}
```

↓Input

will produce the following output:

↑Output

- Animal
- Vegetable
- Mineral

↓Output

It is also possible to nest `itemize` environments. For example, the following code:

↑Input

```
\begin{itemize}
\item Animal
\begin{itemize}
\item Mammals
\item Birds
\item Reptiles. For example:
\begin{itemize}
\item dinosaurs
\item crocodiles
\end{itemize}
\end{itemize}
\item Vegetable
\begin{itemize}
\item Cultivated
```

```
\item Wild
\end{itemize}
\item Mineral
\end{itemize}
```

↓Input

will produce the following output:

↑Output

- Animal
  - Mammals
  - Birds
  - Reptiles. For example:
    - dinosaurs
    - crocodiles
- Vegetable
  - Cultivated
  - Wild
- Mineral

↓Output

That looks good, but our code is a bit cramped and a little difficult to read. Blank lines between list items are ignored by LaTeX, and multiple spaces are treated as a single space, so we could make the code a bit more readable, without affecting the final result:

↑Input

```
\begin{itemize}

   \item Animal

   \begin{itemize}

      \item Mammals

      \item Birds

      \item Reptiles. For example:
      \begin{itemize}

         \item dinosaurs

         \item crocodiles

       \end{itemize}

   \end{itemize}
```

```
   \item Vegetable

   \begin{itemize}

      \item Cultivated

      \item Wild

   \end{itemize}

   \item Mineral

\end{itemize}
```

↓Input

It's now a little easier to see which `\begin{itemize}` matches up with the corresponding `\end{itemize}`.

4.3.2 Ordered Lists

Ordered lists are created using the `enumerate` environment. It has exactly the same format as the `itemize` environment described in the previous section.

We can use the same example as before, only this time use `enumerate` instead of `itemize`.

↑Input

```
\begin{enumerate}
\item Animal
\item Vegetable
\item Mineral
\end{enumerate}
```

↓Input

The above input will produce the following output:

↑Output

1. Animal
2. Vegetable
3. Mineral

↓Output

Again, the environments can be nested:

↑Input

```
\begin{enumerate}

   \item Animal

   \begin{enumerate}
```

```
      \item Mammals

      \item Birds

      \item Reptiles. For example:
      \begin{enumerate}

         \item dinosaurs

         \item crocodiles

       \end{enumerate}

   \end{enumerate}

   \item Vegetable

   \begin{enumerate}

      \item Cultivated

      \item Wild

   \end{enumerate}

   \item Mineral

\end{enumerate}
```

↓Input

The above input will produce the following output:

↑Output

1. Animal
    (a) Mammals
    (b) Birds
    (c) Reptiles. For example:
        i. dinosaurs
        ii. crocodiles
2. Vegetable
    (a) Cultivated
    (b) Wild
3. Mineral

↓Output

4.3.3 Description Environment

The `description` environment has exactly the same format as the `itemize` environment described in Section 4.3.1, only this time you need to specify a label as an optional argument to the `\item` command. For example, the following code:

↑Input

```
\begin{description}
\item[Animal] Living being
\item[Vegetable] Plant
\item[Mineral] Natural inorganic substance
\end{description}
```

↓Input

will produce the following output:

↑Output

Animal Living being

Vegetable Plant

Mineral Natural inorganic substance

↓Output

It is possible to nest all the listing environments:

↑Input

```
\begin{description}

   \item[Animal] Living being

   \begin{itemize}

      \item Mammals

      \item Birds

      \item Reptiles. For example:
      \begin{enumerate}

         \item dinosaurs

         \item crocodiles

       \end{enumerate}

   \end{itemize}

   \item[Vegetable] Plant

   \begin{itemize}

      \item Cultivated
```

```
    \item Wild

  \end{itemize}

  \item[Mineral] Natural inorganic substance

\end{description}
```

↓Input

The above input will produce the output:

↑Output

Animal Living being

- Mammals
- Birds
- Reptiles. For example:
  1. dinosaurs
  2. crocodiles

Vegetable Plant

- Cultivated
- Wild

Mineral Natural inorganic substance

↓Output

Exercise 4 (Lists)

Try writing the source code that will create the following output:

↑Output

Village A small collection of dwelling places. Examples:

1. Marlingford
2. Saxlingham

Town A large collection of dwelling places. Examples:

1. Great Yarmouth
2. Beccles

City A large town, usually containing a cathedral. Examples:

1. Norwich
2. Birmingham
3. London

↓Output

You can download or view the answer if you can't work out how to do it.

4.4 Simple font changing commands

There are two basic ways of changing fonts: you can either change the font for a small selection of text, for example, if you want to *emphasize* a word, or you may wish to change the font "from this point onwards". The commands shown in Table 4.4 are of the first type, whereas those shown in Table 4.5 are of the second type — a declaration.

Table 4.4: Font changing commands

Command	Example Input	Corresponding output
`\textrm{`*text*`}`	`\textrm{roman} text`	roman text
`\textsf{`*text*`}`	`\textsf{sans serif} text`	sans serif text
`\texttt{`*text*`}`	`\texttt{typewriter} text`	`typewriter` text
`\textmd{`*text*`}`	`\textmd{medium} text`	medium text
`\textbf{`*text*`}`	`\textbf{bold} text`	**bold** text
`\textup{`*text*`}`	`\textup{upright} text`	upright text
`\textit{`*text*`}`	`\textit{italic} text`	*italic* text
`\textsl{`*text*`}`	`\textsl{slanted} text`	*slanted* text
`\textsc{`*text*`}`	`\textsc{Small Caps} text`	Small Caps text
`\emph{`*text*`}`	`\emph{emphasized} text`	*emphasized* text
`\textnormal{`*text*`}`	`\textnormal{default} text`	default text

Table 4.5: Font changing declarations

Declaration	Example Input	Corresponding output
`\rmfamily`	`\rmfamily roman text`	roman text
`\sffamily`	`\sffamily sans serif text`	sans serif text
`\ttfamily`	`\ttfamily typewriter text`	`typewriter text`
`\mdseries`	`\mdseries medium text`	medium text
`\bfseries`	`\bfseries bold text`	**bold text**
`\upshape`	`\upshape upright text`	upright text
`\itshape`	`\itshape italic text`	*italic text*
`\slshape`	`\slshape slanted text`	*slanted text*
`\scshape`	`\scshape Small Caps text`	Small Caps text
`\em`	`\em emphasized text`	*emphasized text*
`\normalfont`	`\normalfont default text`	default text

The size of the font is changed using one of the declarations shown in Table 4.6. The sizes are all relative to the size of the normal font. So if you decide to change the normal font from, say, 11pt to 12pt (by changing the class file option as mentioned on page 31), all the font sizes will be changed relative to the new size.

Environments can be used instead. Each environment has the same name as its corresponding declaration, but *without* the preceeding backslash. Example:

↑Input

```
\begin{itshape} Some italic text.
\begin{Large}
This text is large.
```

Table 4.6: Font size changing declarations

Declaration	Example Input	Corresponding output
\tiny	\tiny tiny text	tiny text
\scriptsize	\scriptsize script sized text	script sized text
\footnotesize	\footnotesize footnote sized text	footnote sized text
\small	\small small text	small text
\normalsize	\normalsize normal sized text	normal sized text
\large	\large large text	large text
\Large	\Large even larger	even larger
\LARGE	\LARGE larger still	larger still
\huge	\huge huge	huge
\Huge	\Huge really huge	really huge

```
\end{Large}
\end{itshape} Back to normal.
```

↓Input

Output:

Some italic text. *This text is large.* Back to normal.

Output

Note that the command `\emph`, the declaration `\em` and the environment `em` behave slightly differently to the corresponding `\textit` command, `\itshape` declaration and `itshape` environment. The latter simply use an italic font, whereas the former will toggle between italic and upright. So if the surrounding font is upright then `\emph`, `\em` and `em` will use the italic font, but if the surrounding font is italic, `\emph`, `\em` and `em` will use an upright font. This is particularly useful in abstracts where the abstract font varies between class files. It is recommended that if your intention is to emphasize something, you should use `\emph` etc rather than `\textit` etc.

For more information on using fonts, including using fonts not covered in this document, see *A Guide to LaTeX* [2] or *The LaTeX Companion* [3].

Exercise 5 (Fonts)

Go back to the document you created in Exercise 1 and change the first paragraph to a large bold font and the second paragraph to normal size italic. Emphasize the words “simple” and “short”. (Again, you can download or view the solution.)

Chapter 5

Creating Chapters, Sections etc

Let's go back to the document we created in Exercise 2. In this chapter we shall modify this document step by step until we have a fully fledged document with title, abstract, table of contents, sections etc.

5.1 Author and title information

The term "title page" is used to indicate the author, title and date information that can either appear on the front cover by itself or along the top of the first page of text. In order to do this, you must first specify the information. Once this information has been specified it can then be displayed.

The author, title and date are entered using the commands:

```
\author{author names}
\title{title text}
\date{document date}
```

Definition

These commands only *store* information, they don't actually display anything. Once you have used these commands, you can then display the information using the command:

```
\maketitle
```

Definition

Note that if you don't use the `\date` command, the current date will be inserted. If you want no date to appear, you need to specify an empty argument:

```
\date{}
```

Input

Exercise 6 (Creating Title Pages)

Try editing the document you created in Exercise 2 to include title information. Modifications are illustrated `like this`:

```
\documentclass[a4paper,11pt]{article}
```

†Code

```
\begin{document}

\title{A Simple Document}
\author{Me}

\maketitle

This is a simple \LaTeX\ document.
Here is the first paragraph.

Here is the second paragraph. As you can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today.

\end{document}
```

↓Code

You can download this document.

5.2 Abstract

The `abstract` environment is used to create an abstract for the document. The way in which the abstract is formatted depends on the class file. The `report` class file will put the abstract on a page by itself, some class files will indent the abstract and some will typeset the abstract in italic. Note also that some class files (such as `book` and `letter`) don't have an `abstract` environment. Abstracts traditionally go at the start of the document after the title, so the `abstract` environment should go after the `\maketitle` command.

Exercise 7 (Creating an Abstract)

Try editing your document so that it has an abstract: Modifications are illustrated `like this`:

↑Code

```
\documentclass[a4paper,11pt]{article}

\begin{document}

\title{A Simple Document}
\author{Me}

\maketitle

\begin{abstract}
A brief document to illustrate how to use \LaTeX.
\end{abstract}

This is a simple \LaTeX\ document.
```

```
Here is the first paragraph.

Here is the second paragraph. As you can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today.

\end{document}
```

↓Code

You can download this document.

5.3 Sections, Subsections ...

Chapters, sections, subsections etc can be inserted using the commands:

\part[*short title*]{*title*}
\chapter[*short title*]{*title*}
\section[*short title*]{*title*}
\subsection[*short title*]{*title*}
\subsubsection[*short title*]{*title*}
\paragraph[*short title*]{*title*}
\subparagraph[*short title*]{*title*}

Definition

Note that the availablity of these commands depends on the class file you are using. For example, the `article` class file that we have been using is designed for short articles, so the `\chapter` command is not defined in the `article` class file, whereas it is defined in the `report` class file.

Each of the commands above have a mandatory argument *title* and an optional argument *short title*. The mandatory argument *title* is simply the title of the chapter/section/subsection etc. For example:

```
\section{Introduction}
```

Input

If you are using the `article` class file, the output will look like:

1 Introduction

Output

Note that you don't specify the section number as LaTeX does this automatically. This means that you can insert a new section or chapter or swap sections around or even change a section to a subsection etc, without having to worry about updating all the section numbers.

If you are using a class file that contains chapters as well as sections, the section number will depend on the chapter. So, for example, if the current section is the 4th section of chapter 5, the section number will be 5.4 (Note that if you are using a class file where the section number depends on the chapter number, you must have a `\chapter` command before your first `\section` command, otherwise your section numbers will come out as 0.1, 0.2 etc).

Unnumbered chapters/sections etc are produced by placing an asterisk * after the command name. For example:

```
\chapter*{Acknowledgements}
```
Input

You can switch to appendices using the command

```
\appendix
```
Definition

then continue using `\chapter`, `\section` etc. For example (using the `report` class file):

↑Input
```
\appendix
\chapter{Derivations}
Some derivations.

\chapter{Tables}
Some tables.
```
↓Input

Exercise 8 (Creating Chapters, Sections etc)

Let's try editing our document so that it now has chapters, sections etc. Since the `article` class file doesn't have chapters, let's change to the `report` class. Changes from our previous document are shown `like this`.

↑Code
```
\documentclass[a4paper,11pt]{report}

\begin{document}

\title{A Simple Document}
\author{Me}

\maketitle

\begin{abstract}
A brief document to illustrate how to use \LaTeX.
\end{abstract}

\chapter{Introduction}
\section{The First Section}

This is a simple \LaTeX\ document.
Here is the first paragraph.

\section{The Next Section}

Here is the second paragraph. As you can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today.

\chapter{Another Chapter}

Here's another very interesting chapter.
We're going to put a picture here later.
```

```
\chapter*{Acknowledgements}

I would like to acknowledge all those
very helpful people who have assisted me in my work.

\appendix
\chapter{Tables}
We're going to put some tables here later.

\end{document}
```

↓Code

(You can download a copy of this file if you like, but I would recommend that you try editing the file yourself to give you practice.)

5.4 Creating a Table of Contents

Once you have all your \chapter, \section etc commands, you can create a table of contents with the command

```
\tableofcontents
```

Definition

This command should go where you want your table of contents to appear (usually after \maketitle).

You may recall from the previous section that the sectioning commands all had an optional argument *short title*. If your chapter or section title is particularly long, you can use *short title* to specify a shorter title that should go in the table of contents. The longer title (given by the other argument *title*) will still appear in the section heading in the main part of the document.

LaTeX processes all source code sequentially, so when it first encounters the \tableofcontents command, it doesn't yet know anything about the chapters, sections etc. So the first time the document is LaTeXed the necessary information is written to the .toc file. The subsequent pass reads the information in from the .toc file, and generates the table of contents.

Exercise 9 (Creating a Table of Contents)

Try modifying your document so that it has a table of contents. Modifications from the previous exercise are illustrated like this:

↑Code

```
\documentclass[a4paper,11pt]{report}

\begin{document}

\title{A Simple Document}
\author{Me}

\maketitle

\tableofcontents
```

```
\begin{abstract}
A brief document to illustrate how to use \LaTeX.
\end{abstract}

\chapter{Introduction}

\section{The First Section}

This is a simple \LaTeX\ document.  Here is the first paragraph.

\section{The Next Section}

Here is the second paragraph. As you can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today.

\chapter{Another Chapter}

Here's another very interesting chapter.
We're going to put a picture here later.

\chapter*{Acknowledgements}

I would like to acknowledge all those
very helpful people who have assisted
me in my work.

\appendix
\chapter{Tables}

We're going to put some tables here later.

\end{document}
```

↓Code

If your table of contents doesn't come out right, try LaTeXing it again. (Again, you can download this file.)

5.5 Cross-Referencing

We have already seen that LaTeX takes care of all the numbering for the chapters etc, but what happens if you want to refer to a chapter or section? There's no point leaving LaTeX to automatically generate the section numbers if you have to keep a track of them all, and change all your cross-references every time you add a new section. Fortunately LaTeX provides a way to generate the correct number, all you have to do is label the part of the document you want to reference, and then refer to this label when you want to cross-reference it. LaTeX will then determine the correct number that needs to be inserted at that point.

The first part, labelling the place you want to reference, is done using the command:

`\label{`*string*`}` Definition

The argument *string* should be a unique textual label. This label can be anything you like as long as it is unique, but it is recommended that it isn't too long or you may use up too much memory. People tend to have their own conventions for labelling. I usually start the label with two or three letters that signify what type of thing I'm labelling. For example, if I'm labelling a chapter I'll start with `ch`, if I'm labelling a section I'll start with `sec`. Example:

↑Input
```
\chapter{Introduction}
\label{ch:intro}
```
↓Input

Another example:

↑Input
```
\section{Technical Details}
\label{sec:details}
```
↓Input

Note that the `\label` command doesn't produce any text, it simply assigns a label. You can now refer to that object using the command:

`\ref{`*string*`}` Definition

Example:

↑Input
```
See Section \ref{sec:results} for an anaylsis
of the results.
```
↓Input

It is a typographical convention that you should never start a new line with a number. For example, if you have the text "Chapter 1" the "1" must be on the same line as the "Chapter". We can do this by using an "unbreakable" space, which will put a space but won't allow LaTeX to break the line at that point. This is done using the `~` special character, so the example above should actually be:

↑Input
```
See Section~\ref{sec:results} for an anaylsis
of the results.
```
↓Input

The `\pageref{`*string*`}` command will insert the page number that the label appeared on. Example:

↑Input
```
See Chapter~\ref{ch:def} on
page~\pageref{ch:def} for a list of definitions.
```
↓Input

The label `ch:def` obviously needs to be defined somewhere:

↑Input

```
\chapter{Definitions}
\label{ch:def}
```

↓Input

In fact, I have done this in my source code for this document, so the above example would look like:

See Chapter 2 on page 4 for a list of definitions. Output

It's not just chapters and sections that you can reference, most of the numbers that LaTeX automatically generates can be cross-referenced. The `enumerate` environment described in Section 4.3.2 automatically numbers the items within an ordered list, so it's possible to label list items. For example:

↑Input

```
\begin{enumerate}

   \item\label{itm:edit} Write or edit source code.

   \item Pass source code to the \LaTeX\ application
        (``\LaTeX\ the document'').

      \begin{itemize}

         \item If there are any error messages,
               return to Step~\ref{itm:edit}.

         \item If there are no error messages, a DVI file
               is created, go to Step~\ref{itm:view}.
      \end{itemize}

   \item\label{itm:view} View DVI file to check the result.

\end{enumerate}
```

↓Input

Output:

↑Output

1. Write or edit source code.
2. Pass source code to the LaTeX application ("LaTeX the document").
   - If there are any error messages, return to Step 1.
   - If there are no error messages, a DVI file is created, go to Step 3.
3. View DVI file to check the result.

↓Output

The `\ref` and `\pageref` commands may come before or after the corresponding `\label` command. As with the table of contents, LaTeX first writes out all the cross-referencing information to another file (the auxiliary `.aux` file), and then reads it in the next time, so you will need to LaTeX your document twice to get everything up-to-date.

If the references aren't up-to-date, you will see the following message at the end of the LaTeX run:

```
LaTeX Warning: Label(s) may have changed.
Rerun to get cross-references right.
```

The following warning

```
LaTeX Warning: There were undefined references.
```

means that LaTeX found a reference to a label that does not appear in the auxiliary file. This could mean that it's a new label, and the warning will go away the next time you LaTeX your document, or it could mean that either you've forgotten to define your label with the `\label` command, or you've simply misspelt the label.

Very occasionally, if you have cross-references and a table of contents, you might have to LaTeX your document three times to get everything up to date. Just check to see if the `Label(s) may have changed` warning appears.

If you have an undefined reference, LaTeX will replace the reference number with two question marks `??` in the output. If this happens, check to see if the above warnings have occured.

Exercise 10 (Cross-Referencing)

Try modifying your code so that it has cross-references. Again, changes made from the previous document are illustrated like this:

↑Code

```
\documentclass[a4paper,11pt]{report}

\begin{document}

\title{A Simple Document}
\author{Me}

\maketitle

\tableofcontents

\begin{abstract}
A brief document to illustrate how to use \LaTeX.
\end{abstract}

\chapter{Introduction}
\label{ch:intro}

\section{The First Section}

This is a simple \LaTeX\ document.  Here is the first paragraph.
The next chapter is Chapter~\ref{ch:another}
and is on page~\pageref{ch:another}.
```

```
The next section is Section~\ref{sec:next}.

\section{The Next Section}
\label{sec:next}

Here is the second paragraph. As you can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today.

\chapter{Another Chapter}
\label{ch:another}

Here's another very interesting chapter.
We're going to put a picture here later.
See Chapter~\ref{ch:intro} for an
introduction.

\chapter*{Acknowledgements}

I would like to acknowledge all those
very helpful people who have assisted
me in my work.

\appendix
\chapter{Tables}

We're going to put some tables here later.
\end{document}
```

↓Code

(You can download a copy of this file.)

5.6 Creating a Bibliography

Bibliographies can be created using the `thebibliography` environment. This environment is very similar to the list making environments described in Section 4.3, but instead of `\item` use

`\bibitem[`*label*`]{`*key*`}` Definition

where *key* is a unique keyword that identifies this item. Your keyword can be anything you like, but as with `\label` I would recommend that you use a short memorable keyword. I tend to use the first author's surname followed by the year of publication. Example:

↑Input

```
\begin{thebibliography}{1}
\bibitem{lamport94} ``\LaTeX\ : a document preparation
system'', Leslie Lamport, 2nd edition (updated for
\LaTeX2e), Addison-Wesley (1994).
```

```
\bibitem{kopka95} ``A Guide to \LaTeX2e: document
preparation for beginners and advanced users'',
Helmut Kopka and Patrick W. Daly, Addison-Wesley (1995).

\bibitem{goossens94} ``The \LaTeX\ Companion'',
Michel Goossens, Frank Mittelbach and
Alexander Samarin, Addison-Wesley, (1994).

\end{thebibliography}
```

↓Input

Output:

↑Output

References

[1] "LaTeX : a document preparation system", Leslie Lamport, 2nd edition (updated for LaTeX2e), Addison-Wesley (1994).

[2] "A Guide to LaTeX2e: document preparation for beginners and advanced users", Helmut Kopka and Patrick W. Daly, Addison-Wesley (1995).

[3] "The LaTeX Companion", Michel Goossens, Frank Mittelbach and Alexander Samarin, Addison-Wesley, (1994).

↓Output

You can cite an item in your bibliography with the command

`\cite[`*text*`]{`*key list*`}` Definition

Example:

↑Input

```
For more information about writing bibliographies see
Goossens \emph{et al.}~\cite{goossens94}.
```

↓Input

Output:

For more information about writing bibliographies see Goossens *et al.* [3]. Output

If you want to cite multiple works, use a comma-separated list: Example:

↑Input

```
For more information about writing bibliographies
see~\cite{kopka95,goossens94}.
```

↓Input

Output:

For more information about writing bibliographies see [2, 3]. Output

The optional argument *text* to the `\cite` command can be used to add text to the citation. Example:

```
For more information about writing bibliographies see
Goossens \emph{et al.}~\cite[Chapter~13]{goossens94}.
```

↑Input

↓Input

Output:

↑Output

For more information about writing bibliographies see Goossens *et al.* [3, Chapter 13].

↓Output

The `thebibliography` environment has a mandatory argument:

`\begin{thebibliography}{`*widest entry*`}` Definition

The argument *widest entry* is the widest label in the list of entries. This helps LaTeX to align the references correctly. In the example above, the labels appeared as: [1], [2] and [3], but they can be changed using the optional argument to the `\bibitem` command. In the above example, the labels were all the same width so the argument `{1}` was used (although `{2}` and `{3}` could just have easily been used). Consider the following example:

↑Input

```
\begin{thebibliography}{Goossens 1994}
\bibitem[Lamport 1994]{lamport94} ``\LaTeX\ : a document
preparation system'', Leslie Lamport, 2nd edition
(updated for \LaTeX2e), Addison-Wesley (1994).

\bibitem[Kopka 1995]{kopka95} ``A Guide to \LaTeX2e: document
preparation for beginners and advanced users'', Helmut Kopka
and Patrick W. Daly, Addison-Wesley (1995).

\bibitem[Goossens 1994]{goossens94} ``The \LaTeX\ Companion'',
Michel Goossens, Frank Mittelbach and
Alexander Samarin, Addison-Wesley, (1994).

\end{thebibliography}
```

↓Input

Output:

↑Output

References

[Lamport 1994] "LaTeX : a document preparation system", Leslie Lamport, 2nd edition (updated for LaTeX2e), Addison-Wesley (1994).

[Kopka 1995] "A Guide to LaTeX2e: document preparation for beginners and advanced users", Helmut Kopka and Patrick W. Daly, Addison-Wesley (1995).

[Goossens 1994] "The LaTeX Companion", Michel Goossens, Frank Mittelbach and Alexander Samarin, Addison-Wesley, (1994).

↓Output

In this example, the widest label is [Goossens 1994] so it is chosen to be the argument of the `thebibliography` environment:

```
\begin{thebibliography}{Goossens 1994}
```

Input

There is an application called BibTeX that can be used in conjunction with LaTeX to help generate bibliographies. This document does not cover BibTeX, but if you are interested I would recommend reading *A Guide to LaTeX* [2] or *The LaTeX Companion* [3]. For those of you who want a quick look on-line, the document *Using LaTeX to Write a PhD Thesis*[1] has a section containing a brief introduction to BibTeX.

Exercise 11 (Creating a Bibliography)

Try added the following chapter to your document:

↑Input

```
\chapter{Recommended Reading}

For a basic introduction to \LaTeX\ see Lamport~\cite{lamport94}.
For more detailed information about \LaTeX\ and
associated applications, consult Kopka and Daly~\cite{kopka95}
or Goossens \emph{et al}~\cite{goossens94}.
```

↓Input

and also add the bibliography shown above to the end of your document. You can download or view the solution, but have a go by yourself first. Remember that, as before, you will need to LaTeX the document twice to get the references up-to-date.

5.7 Page Styles and Page Numbering

You may have noticed that the documents you have created have all had their page numbers automatically inserted at the foot of most of the pages. If you have created the document that has gradually been modified over the previous few sections, you may have noticed that the title page has no header or footer, the table of contents is page 1, the abstract page has no page number, and the page after the abstract starts at page 1 and continues incrementally onwards from that point. All the page numbers are Arabic numbers. This can be changed using the command:

```
\pagenumbering{style}
```

Definition

where *style* can be one of:

`arabic`	Arabic page numbers (1, 2, 3, ...)
`roman`	Lowercase Roman numerals (i, ii, iii, ...)
`Roman`	Uppercase Roman numerals (I, II, III, ...)
`alph`	Lower case alphabetical characters (a, b, c, ...)
`Alph`	Upper case alphabetical characters (A, B, C, ...)

[1] http://theoval.cmp.uea.ac.uk/~nlct/latex/thesis/thesis.html

Traditionally, the front matter (table of contents, list of figures etc) should have lowercase Roman numeral page numbering, while the main matter should be in Arabic numerals. Example (using `report` class file):

↑Input

```
\author{Me}
\title{A Simple Document}
\maketitle

\pagenumbering{roman}
\tableofcontents

\begin{abstract}
This is the abstract.
\end{abstract}

\pagenumbering{arabic}
\chapter{Introduction}
```

↓Input

Note that if you don't have an `abstract` environment, you will need to do `\clearpage` before doing `\pagenumbering{arabic}`:

↑Input

```
\author{Me}
\title{A Simple Document}
\maketitle

\pagenumbering{roman}
\tableofcontents

\clearpage\pagenumbering{arabic}
\chapter{Introduction}
```

↓Input

The headers and footers can be changed using the command

`\pagestyle{`*style*`}` Definition

Individual pages can be changed using

`\thispagestyle{`*style*`}` Definition

Standard styles are:

`empty`	No header or footer.
`plain`	Header empty, page number in footer.
`headings`	Header contains page number and various information, footer empty.
`myheadings`	Header specified by user, footer empty.

If the `myheadings` style is used, the header information can be specified using:

`\markboth{`*left head*`}{`*right head*`}` Definition

if the `twoside` option has been passed to the class file, or

`\markright{`*right head*`}` Definition

if the `oneside` option has been passed to the class file (default for `article` and `report`).

The `report` class file uses the `empty` style for the title and abstract pages and `plain` for the first page of each new chapter. By default the remaining pages are also `plain`, but these can be changed using the `\pagestyle` command. This document uses the `headings` page style. As you can see the chapter number and title appear in the top left and the page number appears in the top right of most pages. The default `oneside` option was used, so there is no difference between the formatting of odd and even numbered pages.

The on-screen PDF version of this document uses a page style I defined myself that incorporates a navigation bar in the footer. (For information on how to do this, see *Creating a PDF Document using PDFLaTeX*[2].)

Exercise 12 (Page Styles and Page Numbering)

Try editing your document so that the page numbering is lowercase Roman for the table of contents but Arabic for the main matter. You can try changing the page style as well, but since the chapters are less than a page each, you won't see any effect until we make our chapters a bit bigger. (You can download or view the edited document.)

5.8 Aligning Material in Rows and Columns

Text can be aligned in rows and columns using the `tabular` environment.

`\begin{tabular}[`*placement specifier*`]{`*column specifiers*`}` Definition

This environment has a mandatory argument *column specifiers* that specifies how to align each column. There are three basic specifiers: `r` (right aligned), `l` (left aligned) and `c` (centred). For example, suppose we want three columns with the first column left justified and the last two columns centred we would do:

```
\begin{tabular}{lcc}
```
Input

(Make sure you don't confuse `l` ('ell') with `1` (one).)

The ampersand character `&` is used to separate column entries and `\\` is used to separate rows. For example, let's have two columns, the first left justified and the second right justified:

↑Input

```
\begin{tabular}{lr}
Video & 8.99\\
CD & 9.99\\
DVD & 15.00\\
Total & 33.98
\end{tabular}
```

↓Input

Output:

↑Output

Video 8.99
CD 9.99
DVD 15.00
Total 33.98

↓Output

Remember that LaTeX ignores multiple spaces, so we could just have easily done:

↑Input

```
\begin{tabular}{lr}
Video & 8.99\\
CD    & 9.99\\
DVD   & 15.00\\
Total & 33.98
\end{tabular}
```

↓Input

and we would still have got the same result.

Entries form implicit grouping, so declarations made within a `tabular` environment only have an effect up to the next `&` or `\\`. Example:

↑Input

```
\begin{tabular}{lr}
Video & 8.99\\
CD    & 9.99\\
DVD   & 15.00\\
\bfseries Total & 33.98
\end{tabular}
```

↓Input

Output:

↑Output

Video 8.99
CD 9.99
DVD 15.00
Total 33.98

↓Output

Let's add an extra column and a header row:

↑Input

```
\begin{tabular}{lrr}
Item  & ex VAT & inc VAT\\
Video & 8.99   & 10.56\\
CD    & 9.99   & 11.74\\
DVD   & 15.00  & 17.63\\
\bfseries Total & 33.98 & 39.93
\end{tabular}
```

↓Input

Output:

↑Output

Item	ex VAT	inc VAT
Video	8.99	10.56
CD	9.99	11.74
DVD	15.00	17.63
Total	33.98	39.93

↓Output

The command

`\multicolumn{`*cols spanned*`}{`*col specifier*`}{`*text*`}` Definition

can be used to span several columns. The first argument *cols spanned* is the number of columns you want to span, the second argument *col specifier* indicates how to align this column spanning entry, the third argument *text* indicates what should go in this entry. We can use `\multicolumn` to modify the previous example as follows:

↑Input

```
\begin{tabular}{lrr}
      & \multicolumn{2}{c}{Price (\pounds)}\\
Item  & ex VAT & inc VAT\\
Video & 8.99   & 10.56\\
CD    & 9.99   & 11.74\\
DVD   & 15.00  & 17.63\\
\bfseries Total & 33.98 & 39.93
\end{tabular}
```

↓Input

Output:

↑Output

	Price (£)	
Item	ex VAT	inc VAT
Video	8.99	10.56
CD	9.99	11.74
DVD	15.00	17.63
Total	33.98	39.93

↓Output

In this example we are spanning two columns, so the first argument to `\multicolumn` is `{2}`, we want the entry centred, so the second argument is `{c}` and the text to go in this entry is simply {`Price (\pounds)`}.

The `\multicolumn` command can also be used to override the alignment of individual entries. Consider the following example:

↑Input

```
\begin{tabular}{lrr}
          & Year1   & Year2 \\
Travel    & 100,000 & 110,000\\
```

[2] http://theoval.cmp.uea.ac.uk/~nlct/latex/pdfdoc/

```
Equipment & 50,000  & 60,000
\end{tabular}
```

↓Input

Output:

↑Output

	Year1	Year2
Travel	100,000	110,000
Equipment	50,000	60,000

↓Output

In this example, the headers 'Year1' and 'Year2' would look better centred, but the rest of the entries in the second and third columns look best right aligned. We can use `\multicolumn` to span just one column, and use the second argument of `\multicolumn` to override the column specification:

↑Input

```
\begin{tabular}{lrr}
          & \multicolumn{1}{c}{Year1}
          & \multicolumn{1}{c}{Year2} \\
Travel    & 100,000 & 110,000\\
Equipment & 50,000  & 60,000
\end{tabular}
```

↓Input

Output:

↑Output

	Year1	Year2
Travel	100,000	110,000
Equipment	50,000	60,000

↓Output

Exercise 13 (Aligning Material)

You may have noticed that the document you have been creating throughout this chapter has an appendix entitled "Tables". The `tabular` environment does not create a table, but later on in Section 7.2 we'll see how to turn it into one. For now, try placing the following `tabular` environment into the appendix of your document:

↑Output

	Expenditure	
	Year1	Year2
Travel	100,000	110,000
Equipment	50,000	60,000

↓Output

You can download or view the result.

For more information about using the `tabular` environment, including how to add vertical and horizontal lines, see the LaTeX user's guide [1], *A Guide to LaTeX* [2]

or *The LaTeX Companion* [3]. The latter reference also describes how to span rows using the `multirow` package. For information on how to create coloured tables using the `colortbl` package, see *The LaTeX Graphics Companion* [4].

Chapter 6

Packages

Packages are files with the extension `.sty` that either define new commands or redefine existing commands. We shall first look at how to use packages already installed on your system, and then we shall look at how to download and install new packages.

6.1 Using Packages

LaTeX has a great many useful commands, but it doesn't have a command to do absolutely everything, so if additional commands are required, they can be supplied in files called packages. If you want to use any commands or environments that are defined in a package, you first need to specify the name of the package with the command:

`\usepackage[`*options*`]{`*package name*`}` Definition

where *package name* is the name of the package without the `.sty` extention, and *options* is a comma separated list of options to be passed to the package (just as you can do with class files using the `\documentclass` command.) Note that the `\usepackage` command must *always* go in the preamble.

Let's look at a couple of examples.

6.1.1 `graphicx` Package

It is possible to generate images using LaTeX commands (See *The LaTeX Graphics Companion* [4]) however most people find it easier to create a picture in some other application, and include that file into their LaTeX document.

Some applications have an option that allows you to save an image as a PostScript file[1]. The `graphicx` package provides a command that enables you to include this PostScript file into your document[2].

Firstly, you need to specify that you want to use the `graphicx` package. So you will need to place the following command in the preamble:

```
\usepackage{graphicx}
```
Input

[1] if it doesn't have this option you can use a PostScript printer driver and print to file. You can also convert other file types to PostScript using applications such as: `pdftops`, `tiff2ps`, `pnmtops`

[2] You can also use other file types, such as `.pdf` or `.png`, depending on what system you are using

The PostScript file can then be included in your document using the command

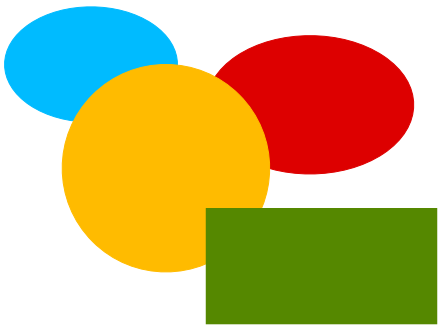
```
\includegraphics[key vals]{filename}
```
Definition

where *filename* is the name of your PostScript file, and *key vals* is a comma-separated list of options that can be used to manipulate the image.

Example: suppose you had a file called `shapes.ps`, then to include it in your document you would do:

```
\includegraphics{shapes.ps}
```
Input

Output:



Output

If you omit the file extension, LaTeX will search for a file with the default extension. If you are using ordinary LaTeX, this will usually be `.ps`, however if you are using PDFLaTeX, this will usually be `.pdf`. Modifying the above example, we could do:

```
\includegraphics{shapes}
```
Input

If we use LaTeX, the file `shapes.ps` will be used, and if we use PDFLaTeX, the file `shapes.pdf` will be used. So, if you sometimes use LaTeX and sometimes use PDFLaTeX, you may find it easier to omit the extension, and have two copies of the image in both PostScript and PDF format.

You can specify which file types to look for with the command

```
\DeclareGraphicsExtensions{ext-list}
```
Definition

where *ext-list* is a comma-separated list of extensions. For example, if you are using PDFLaTeX, you might want to search first for PDF files, and then for PNG files:

```
\DeclareGraphicsExtensions{.pdf,.png}
```
Input

or if you are using LaTeX and `dvips`, you might want to first search for encapsulated PostScript (EPS) files and then for PostScript (PS) files:

```
\DeclareGraphicsExtensions{.eps,.ps}
```
Input

The optional argument *key vals* should be a comma separated list of *key=label* pairs. Common options are:

`angle=`x	rotate the picture by x°
`width=`*len*	scale the picture so that the width is *len*. (Remember to specify the units)
`height=`*len*	scale the picture so that the height is *len*. (Remember to specify the units)
`scale=`*value*	Scale the picture by *value*
`trim=`*l b r t*	Specifies the amount to remove from each side. E.g. `trim=1 2 3 4` crops the picture by 1bp from the left, 2bp from the bottom, 3bp from the right and 4bp from the top. (The unit `bp` is a PostScript point 72bp = 1in)
`draft`	Don't actually print the image, just draw a box of the same size and print the filename inside it.

Let's try rotating and scaling our picture:

```
\includegraphics[angle=45,width=1in]{shapes}
```
Input

Output:

Output

The `graphicx` package also provides commands to rotate, resize, reflect and scale text. They are as follows:

- `\rotatebox{`*angle*`}{`*text*`}`
  Example:

  ```
  \rotatebox{45}{Some text}
  ```
  Input

  Output:

  Output

- `\scalebox{`*h scale*`}[`*v scale*`]{`*text*`}`
  Example:

  ```
  \scalebox{0.8}{Some text}
  ```
  Input

Output:

Some text Output

- `\reflectbox{`*text*`}`
  Example:

```
\reflectbox{Some text}
```
Input

Output:

Some text Output

- `\resizebox{`*h length*`}{`*v length*`}{`*text*`}`
  Example:

```
\resizebox{12mm}{1cm}{Some text}
```
Input

Output:

Output

The `graphicx` package can have the following options passed to it:

`draft` Don't actually display the images, just print the filename in a box of the correct size. This is useful if you want to print out a draft copy of a document to check the text rather than the images.

`final` Opposite of `draft` (default).

`hiderotate` Don't show rotated text.

`hidescale` Don't show scaled text.

Example:

```
\usepackage[draft]{graphicx}
```
Input

Exercise 14 (Using the `graphicx` Package)

Download the file shapes.ps, and include it into your document. Alternatively, if you prefer to use PDFLaTeX, you can download the file shapes.pdf instead. (You can download or view an example solution.)

Some previewers may not be able to display PostScript images or perform the scaling, rotating etc, in this case you can use `dvips` to convert your DVI file into a PostScript file either using the MS-DOS Prompt, WinEdt or TeXnicCenter, as described in Chapter 3, and then view it using `GSView`.

For more information on the `graphicx` package see *The LaTeX Graphics Companion* [4].

6.1.2 Changing the format of `\today`

In the document we have been creating in the exercises, we have used the command `\today` to produce the current date. By default, this command displays the date in a US format, e.g. September 27, 2004, but you might prefer a UK format. This can be done by loading a package that redefines the `\today` command. There are several packages available, amongst which are: `ukdate` and `datetime`.

For example, if you want to use the `ukdate` package, you would type the following in the preamble:

```
\usepackage{ukdate}
```
Input

and the command `\today` will then display the date in the form: Monday 27th September, 2004

The `datetime` package provides twelve different date formats, as well as providing commands for printing the current time. The required date format can either be set using a declaration, or by passing the relevant option to the package. For example, to redefine `\today` to display the date in the form 27/09/2004, you can either do

```
\usepackage[ddmmyyyy]{datetime}
```
Input

or

```
\usepackage{datetime}
\ddmmyyyydate
```
↑Input ↓Input

The `datetime` package also provides a command to define your own date format (and your own time format) if the available formats don't meet your requirements.

6.2 Downloading and Installing Packages

New LaTeX packages are being created all the time, so you may find that there are some packages that you don't have on your installation. In this case, if you don't have the package you want, you can download it from the TeX Archive [6].

Many packages are supplied with the code and documentation all bundled together in one file. This file usually has the extension `.dtx`, and it usually comes with an installation script that has the extension `.ins`. Once you have downloaded the `.dtx` and `.ins` files, you will then have to extract the code before you can use it. Let's illustrate this with an example.

The datetime package comes with the files datetime.dtx and datetime.ins. You need to download both these files. You then need to LaTeX the installation script to obtain the file datetime.sty. You do this in the MS-DOS Prompt by typing:

```
latex datetime.ins
```

This should create the file datetime.sty. Now you need to extract the documentation which details what commands are supplied with this package. You do this by LaTeXing the file datetime.dtx:

```
latex datetime.dtx
```

This should create the file datetime.dvi. Alternatively, you can use PDFLaTeX:

```
pdflatex datetime.dtx
```

This will obtain the file datetime.pdf, and since the file datetime.dtx uses the hyperref package, all the cross-references in the PDF document will be active links.

The file datetime.sty needs to be put somewhere where LaTeX can find it. LaTeX usually searches subdirectories of c:\texmf\tex\latex, depending on how your system is configured. New files should be placed in the local texmf directory tree, which is usually c:\localtexmf\tex\latex. Put the file datetime.sty in a subdirectory of c:\localtexmf\tex\latex[3] (e.g. c:\localtexmf\tex\latex\datetime), and place datetime.dvi or datetime.pdf in one of the documentation subdirectories (usually in c:\localtexmf\doc\latex). Once you have done this you will need to update the TeX database. To do this you need to run MiKTeX Options which will probably be in:

Start → Programs → MiKTeX → MiKTeX Options

and then click on the Refresh Now button (See Figure 6.1). If you are using UNIX or Linux you need to use the command texhash or mktexlsr. Recent versions of MiKTeX have an application called MiKTeX Update Wizard which can automatically download and install known packages, check the MiKTeX documentation for further details.

If you experience any problems, contact your system administrator for help.

Alternatively, you can leave the .sty file in the same directory as your LaTeX document, but if you do this, you will only be able to use it with documents in that directory.

As mentioned in the previous section, the datetime package has various options that can be used to change the format of \today. For example, by default the datetime package redefines \today to display the date in the form: Monday 27th September, 2004. The option short will produce an abbreviated form, (e.g. Mon 27th Sept, 2004) and the option nodayofweek won't display the day of the week (e.g. 27th September, 2004). These can be passed as a comma separated list in the optional argument to the \usepackage command. The datetime package (version 2.3 and above) also defines the command \currenttime which displays the current time. For example:

↑Input

```
\documentclass[a4paper,11pt]{article}

\usepackage[short,nodayofweek]{datetime}

\begin{document}
```

[3] you may have to create the relevant subdirectories

Figure 6.1: Updating the database

```
This is a simple \LaTeX\ document.
Here is the first paragraph.

Here is the second paragraph. As you
can see it's a very
short document\footnote{with a footnote}.
This document was created on: \today\ at
\currenttime.

\end{document}
```

↓Input

Exercise 15 (Downloading and Installing a New Package)

Try downloading the `datetime` package to give you practice extracting and installing packages. Then edit your document so that it uses the `datetime` package. (You can download or view an example.)

Chapter 7

Figures and Tables

Figures and tables are refered to as "floats" because they are *floated* to the nearest location. Floats have a caption and associated number. It is customary for figure captions to appear at the bottom of the figure and for table captions to appear at the top of the table. Figures and tables may not have page breaks within them (although there is a package called `longtable` that allows you to have a table that spans several pages, but that's not covered here.)

7.1 Figures

Figures are created using the `figure` environment. The command:

`\caption[`*short caption*`]{`*text*`}` Definition

is used to generate the caption.

Recall from Section 6.1.1 we can include a PostScript or PDF image in our document with the command `\includegraphics` defined in the `graphicx` package. We can put our `shapes.ps` or `shapes.pdf` image into a figure as follows:

↑Input
```
\begin{figure}
\includegraphics{shapes}
\caption{Some shapes}
\end{figure}
```
↓Input

So far so good, but our picture needs to be centred. This can be done using the command `\centerline{`*object*`}`:

↑Input
```
\begin{figure}
\centerline{\includegraphics{shapes}}
\caption{Some shapes}
\end{figure}
```
↓Input

The `\caption` command generates a number, just like `\section`, so we can cross-reference it with `\ref` and `\label`. First, let's label the figure:

↑Input

```
\begin{figure}
\centerline{\includegraphics{shapes}}
\caption{Some shapes}
\label{fig:shapes}
\end{figure}
```

↓Input

Now we can reference it:

```
Figure~\ref{fig:shapes} shows some shapes.
```

Input

(As before we use ~ to make an unbreakable space.) This produces the following output:

Figure 7.1 shows some shapes.

Output

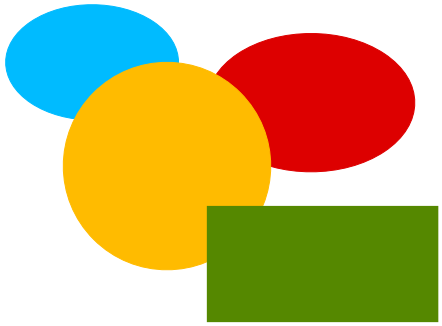


Figure 7.1: Some shapes

Just as we were able to generate a table of contents using `\tableofcontents`, we can also generate a list of figures using the command

```
\listoffigures
```

Definition

As before you will need to LaTeX your document twice to get the list of figures up-to-date.

Exercise 16 (Creating Figures)

If you did Exercise 14, you should have a document with the image `shapes.ps` (or `shapes.pdf`) in it. You now need to put this image into a `figure` environment. Remember to centre the image, and give the figure a caption. Next, try labelling the figure and referencing it in the text. You could also put in a list of figures after the table of contents.

(You can download or view an example.)

Coming up next is a description of the `subfigure` package. If you're struggling a bit you can skip this bit and move on to Section 7.2 on page 74.

7.1.1 Subfigures

Some figures have subfigures within them. These can be generated using the `subfigure` package. Each subfigure is specified using

`\subfigure[`*caption*`]{`*object*`}` Definition

For example, suppose you have two files `circle.ps` and `rectangle.ps`:

↑Input
```
\begin{figure}
\begin{center}
\subfigure[A Rectangle]{\includegraphics{rectangle.ps}}
\hspace{1in}
\subfigure[A Circle]{\includegraphics{circle.ps}}
\end{center}
\caption{Two Shapes: (a) A Rectangle and (b) A Circle}
\end{figure}
```
↓Input

A few notes:

- `\hspace{`*len*`}` make a horizontal space of length *len*.
- The `center` environment centres its contents.

Again we can cross-reference the subfigures. The `\label` command should go in the mandatory argument of the `\subfigure` command.

↑Input
```
\begin{figure}
\begin{center}
\subfigure[A Rectangle]{%
\label{fig:rectangle}\includegraphics{rectangle.ps}}
\hspace{1in}
\subfigure[A Circle]{%
\label{fig:circle}\includegraphics{circle.ps}}
\end{center}
\caption{Two Shapes: (a) A Rectangle and (b) A Circle}
\label{fig:shapes2}
\end{figure}

Figure~\ref{fig:shapes2} shows some shapes.
Figure~\ref{fig:rectangle} shows a rectangle and
Figure~\ref{fig:circle} shows a circle.
```
↓Input

This produces the following output:

↑Output

Figure 7.2 shows some shapes. Figure 7.2(a) shows a rectangle and Figure 7.2(b) shows a circle.

↓Output

and produces Figure 7.2.

(a) A Rectangle (b) A Circle

Figure 7.2: Two Shapes: (a) A Rectangle and (b) A Circle

Exercise 17 (Creating Sub-Figures)

Download `rectangle.ps` and `circle.ps` (or `rectangle.pdf` and `circle.pdf`) and add Figure 7.2 to your document. You can download or view an example.

7.2 Tables

Tables are produced in much the same way as figures, except that the `table` environment is used instead. It is a typographical convention to have the caption at the top of the table (as opposed to figures, which have the caption at the bottom). Example:

↑Input

```
\begin{table}
\caption{A Sample Table}
\label{tab:sample}
\centerline{
\begin{tabular}{lr}
Item & Cost\\
Video & 8.99\\
CD    & 9.99\\
DVD   & 15.00
\end{tabular}}
\end{table}
```

↓Input

This produces Table 7.1.

Table 7.1: A Sample Table

Item	Cost
Video	8.99
CD	9.99
DVD	15.00

Again, `\centerline` is used to centre the `tabular` environment, however the table is a little cramped, so let's put in a bit of extra vertical space after the caption. This can be done using the command:

Table 7.2: A Sample Table

Item	Cost
Video	8.99
CD	9.99
DVD	15.00

`\vspace{`*length*`}` Definition

Our code now looks like:

↑Input

```
\begin{table}
\caption{A Sample Table}
\label{tab:sample}
\vspace{10pt}
\centerline{
\begin{tabular}{lr}
Item & Cost\\
Video & 8.99\\
CD    & 9.99\\
DVD   & 15.00
\end{tabular}}
\end{table}
```

↓Input

This produces Table 7.2.

As with figures, you can create a list of tables using the command

`\listoftables` Definition

Exercise 18 (Creating Tables)

If you did Exercise 13, you should have a `tabular` environment in your document. Try turning this into a table, and add Table 7.2. You could also try adding a list of tables. You can download or view the document.

Chapter 8

Defining Commands

It is possible to define your own commands or redefine existing ones. Be very careful about redefining existing commands; don't redefine a command simply because you want to use the name, only redefine it if you are making a modification. For example, if you want to change the format of the current date, you would redefine `\today`, but if you want to define a command to display a specific date, you should define a new command with a different name.

There are several reasons why you might want to define a new command:

1. Reduce typing:

   Suppose you have a series of commands or text that you find yourself frequently using, then you could define a command to do all these other commands for you.

   Example: Suppose you want a lot of large bold slanted sans-serif portions of text within your document. Every time you type those portions of text, you will have to do something like:

   ```
   \textsf{\large\bfseries\slshape Some text}
   ```
   Input

   It would be much easier if you could use just one command to do all that, called, say, `\largeboldsfsl`:

   ```
   \largeboldsfsl{Some text}
   ```
   Input

   or we could call it, say, `\lbsfsl` which is shorter, but slightly less memorable:

   ```
   \lbsfsl{Some text}
   ```
   Input

2. Ensure consistency:

   You may find that you want to format an object a certain way. For example, your document may have a lot of keywords in it, and you may want to format these keywords in a different font, say sans-serif, so that they stand out. You could just do:

```
A \textsf{command} usually begins with a backslash.
```
Input

however, it is better to define a new command called, say, `\keyword` that will typeset its argument in a sans-serif font. That way it becomes a lot easier to change the format at some later date. For example, you may decide to splash out and have your keywords typed in a particular colour. In which case, all you need to do is simply change the definition of the command `\keyword`, otherwise you'll have to go through your entire document looking for keywords and changing each one which could be very time consuming if you have a large document. You might also decide at some later date to make an index for your document. Indexing all the keywords then becomes very simple, as again all you'll need to do is modify the `\keyword` command.

New commands are defined using the command:

`\newcommand{`*cmd*`}[`*n-args*`][`*default*`]{`*text*`}`
Definition

The first mandatory argument *cmd* is the name of your new command, which must start with a backslash. The optional argument *n-args* specifies how many arguments your new command must take. The next optional argument *default* will be discussed later. The final mandatory argument *text* specifies what LaTeX should do everytime it encounters this command.

Let's begin with a trivial example. Suppose I wanted to write a document about a particular course, say "Programming — Languages and Software Construction", and I had to keep writing the course title, then I might decide to define a command that prints the course title rather than having to laboriously type it out every time. Let's call our new command `\coursetitle`. We want the following code:

```
The course \emph{\coursetitle} is an undergraduate course.
```
Input

to produce the following output:

↑Output

The course *Programming — Languages and Software Construction* is an undergraduate course.

↓Output

Clearly this command doesn't need any arguments, so we don't need to worry about the optional argument *n-args* to `\newcommand`, and the only thing our new command needs to do is print:
`Programming --- Languages and Software Construction`
so we would define our new command as follows:

↑Input

```
\newcommand{\coursetitle}{Programming --- Languages
and Software Construction}
```

↓Input

Commands must always be defined before they are used. The best place to define commands is in the preamble:

↑Code

```
\documentclass[a4paper]{article}

\newcommand{\coursetitle}{Programming --- Languages
and Software Construction}

\begin{document}

\section{\coursetitle}

The course \emph{\coursetitle} is an undergraduate course.

\end{document}
```

↓Code

Now let's try defining a command that takes an argument (or parameter). Let's go back to our `\keyword` example. This command needs to take one argument that is the keyword. Let's suppose we want keywords to come out in sans-serif, then we could do:

```
\newcommand{\keyword}[1]{\textsf{#1}}
```

Input

In this case we have used the optional argument *n-args* to `\newcommand`. We want our command `\keyword` to have one argument, so we have `[1]`. In `\textsf{#1}` the `#1` represents the first argument. (If we had more than one argument, `#2` would represent the second argument, `#3` would respresent the third argument etc. up to a maximum of 9.) So

```
\keyword{commands}
```

will be equivalent to

```
\textsf{commands}
```

and

```
\keyword{environment}
```

will be equivalent to

```
\textsf{environment}
```

and so on.

Again, the line

```
\newcommand{\keyword}[1]{\textsf{#1}}
```

should go in the preamble. That way you can ensure the command won't be used before it's defined:

↑Code

```
\documentclass[a4paper]{article}

\newcommand{\keyword}[1]{\textsf{#1}}

\begin{document}

A \keyword{command} usually begins with a backslash.

\end{document}
```

↓Code

Now if we want to change the way the keywords are formatted, we can simply change the definition of `\keyword`. Let's modify our code so that the keyword is now in a slanted sans-serif font:

↑Code

```
\documentclass[a4paper]{article}

\newcommand{\keyword}[1]{\textsf{\slshape #1}}

\begin{document}

A \keyword{command} usually begins with a backslash.

\end{document}
```

↓Code

Let's go one stage further. The `color` package enables the use of colour, so let's make our keywords blue:

↑Code

```
\documentclass[a4paper]{article}

\usepackage{color}

\newcommand{\keyword}[1]{\textsf{\slshape\color{blue}#1}}

\begin{document}

A \keyword{command} usually begins with a backslash.

\end{document}
```

↓Code

Or we could index the keywords. To do this we need the package `makeidx` and the commands `\makeindex`, `\index{`*text*`}` and `\printindex`:

↑Code

```
\documentclass[a4paper]{article}
```

```
\usepackage{makeidx}

\makeindex

\newcommand{\keyword}[1]{\textsf{\slshape #1}\index{#1}}

\begin{document}

A \keyword{command} usually begins with a backslash.

\printindex

\end{document}
```

↓Code

For further information about how to create an index, see *A Guide to LaTeX* [2] or *The LaTeX Companion* [3]. Alternatively, if you want a brief overview on-line, try *Using LaTeX to Write a PhD Thesis*[1].

Exercise 19 (Defining a New Command)

Try typing up the following code:

↑Code

```
\documentclass[a4paper]{article}

\newcommand{\keyword}[1]{\textsf{#1}}

\begin{document}

A \keyword{command} usually begins with a backslash.

Segments of code may be \keyword{grouped}.

Some \keyword{commands} take one or more \keyword{arguments}.

\end{document}
```

↓Code

Then modify your code so that the keywords are in a slanted sans-serif font, and then modify your code so that the keywords come out in blue. (You may need to convert your DVI file to PostScript in order to see the colour, using `dvips` as described in Chapter 3, or use PDFLaTeX instead of LaTeX.) Again you can download or view the result.

For the more adventurous:
If you want to create an index as in the previous example, you will need to use the application `makeindex`. Suppose your source code is saved as `exercise19.tex`, then if you are using the `MS-DOS Prompt` you will need to do:

[1] http://theoval.cmp.uea.ac.uk/~nlct/latex/thesis/thesis.html

```
latex exercise19.tex
makeindex exercise19.idx
latex exercise19.tex
```

If you are using WinEdt click the LaTeX button, then select Makeindex from the menu, then click on the LaTeX button again. If you are using TeXnicCenter, if you select 'uses MakeIndex' when you create your project, TeXnicCenter will automatically call makeindex when you click on the build icon. If you have already created the project, you can modify its settings using the Project menu.

8.1 Defining Commands with an Optional Argument

As mentioned earlier, the \newcommand command has an optional argument *default*. This allows you to define a command with an optional argument. For example, suppose we want a command called, say, \price. Suppose we want the following code:

```
\price{100}
```
Input

to produce the following output:

£100 excl VAT @ 17.5%

Output

and let's suppose we want an optional argument so that we can change the VAT. That is, we would want the following code:

```
\price[0]{30}
```
Input

produce the following output:

£30 excl VAT @ 0%

Output

Therefore we want to define a command such that if the optional argument is absent we will have 17.5, and if it is present the optional argument will be substituted instead. This command can be defined as follows:

```
\newcommand{\price}[2][17.5]{\pounds #2 excl VAT @ #1\%}
```
Input

Here, #1 represents the optional argument (by default 17.5) and #2 represents the mandatory argument (the second argument if the optional argument is present, or the only argument if the optional argument is absent.)

Exercise 20 (Defining Commands with an Optional Argument)

In this exercise, you will need to define a slightly modified version of the above example. Try defining a command called, say, `\cost`. It should take one optional argument and one mandatory argument. Without the optional argument, it behaves in the same way as the `\price` example above, so that, say,

```
\cost{50}
```
Input

will produce

£50 excl VAT @ 17.5%
Output

but with the optional argument, you can change the `excl VAT @ 17.5\%` bit. So that, say,

```
\cost[inc VAT]{50}
```
Input

will produce

£50 inc VAT
Output

You can download or view the solution.

8.2 Redefining Commands

Commands can be redefined using the command:

`\renewcommand{`*cmd*`}[`*n-args*`][`*default*`]{`*text*`}`
Definition

This has exactly the same format as `\newcommand` but is used for redefining existing commands. **Caveat:** never redefine a command whose existing function is unknown to you.

Recall the `itemize` environment from Section 4.3.1. You may have up to four nested `itemize` environments, the labels for the outer environment are specified by the command `\labelitemi`, the labels for the second level are specified by `\labelitemii`, the third by `\labelitemiii` and the fourth by `\labelitemiv`. By default, `\labelitemi` is a bullet point, `\labelitemii` is an en dash, `\labelitemiii` is an asterisk and `\labelitemiv` is a dot (• – ∗ ·). These can be changed by redefining `\labelitemi` etc.

Example: Recall the command `\dag` produces a dagger symbol, we can use this symbol instead of a bullet point:

```
\renewcommand{\labelitemi}{\dag}

\begin{itemize}
```
Input

```
\item Animal

\item Mineral

\item Vegetable

\end{itemize}
```

↓Input

Output:

↑Output

- † Animal
- † Mineral
- † Vegetable

↓Output

Here's another example, it uses the PostScript font[2] ZapfDingbats via the `pifont` package:

↑Input

```
\renewcommand{\labelitemi}{\ding{43}}

\begin{itemize}

\item Animal

\item Mineral

\item Vegetable

\end{itemize}
```

↓Input

Output:

↑Output

- ☞ Animal
- ☞ Mineral
- ☞ Vegetable

↓Output

In the above example, it would actually be better to use the `dinglist` environment defined in the `pifont` package. See *The LaTeX Companion* [3] for more details.

You may have noticed that LaTeX automatically generates pieces of text such as "Chapter", "Figure", "Bibliography". These are generated by the commands shown in Table 8.1.

[2] This font may not come out if you view the PDF version of your document in `xpdf`

Table 8.1: Object Names (†report class file, ‡article class file, remainder both report and article)

Command	Default Text
\contentsname	Contents
\listfigurename	List of Figures
\listtablename	List of Tables
\bibname†	Bibliography
\refname‡	References
\indexname	Index
\figurename	Figure
\tablename	Table
\partname	Part
\chaptername†	Chapter
\appendixname	Appendix
\abstractname	Abstract

You can change the defaults using `\renewcommand`. For example, suppose you want the table of contents to be labelled "Table of Contents", instead of the default "Contents", you would need to do:

```
\renewcommand{\contentsname}{Table of Contents}
```
Input

Exercise 21 (Renewing Commands)

If you did Exercises 16 and 18, go back to that document and changed the figures and tables so that they are labelled "Fig" and "Tab" instead of "Figure" and "Table".

You can download or view the solution.

Chapter 9

Mathematics

As mentioned in the Introduction, LaTeX is particularly good at typesetting mathematics. In order to use any of the maths commands we need to be in one of the mathematics environments. There are two basic types of mathematics: in-line maths and displayed maths. In-line maths is mathematics that occurs within a line of text, for example:

The variable x is transformed by the function $f(x)$. Output

Displayed maths is mathematics that occurs on a line of its own. For example:

↑Output

A polynomial is a function of the form

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

↓Output

9.1 In-Line Mathematics

In-line mathematics is created using the `math` environment. (Note U.S. spelling — 'math' not 'maths'). Example:

↑Input

```
The variable \begin{math}x\end{math} is transformed
by the function \begin{math}f(x)\end{math}.
```

↓Input

It's somewhat cumbersome having to type `\begin{math}` and `\end{math}` and it also makes the source code a little difficult to read so there are shorthand notations that can be used instead: `\(` is equivalent to `\begin{math}` and `\)` is equivalent to `\end{math}`. So the example above can be rewritten:

↑Input

```
The variable \(x\) is transformed
by the function \(f(x)\).
```

↓Input

There is an even shorter notation: The special character `$` is equivalent to both `\begin{math}` and `\end{math}`:

↑Input

```
The variable $x$ is transformed
by the function $f(x)$.
```

↓Input

This is considerably easier to type and to read, but you need to make sure that all your `$` symbols have matching pairs. The above code will look like:

The variable x is transformed by the function $f(x)$. Output

Note: you should always make sure you are in maths mode to typeset any variables (such as x, y, z), as this will ensure that the correct maths fonts are used.

↑Input

```
Notice the difference between $(x, y, z)$ and
\textit{(x, y, z)}.
```

↓Input

Notice the difference between (x, y, z) and *(x, y, z)*. Output

9.2 Displayed Mathematics

Displayed mathematics can be created using either the `displaymath` or the `equation` environments. Example:

↑Input

```
A linear function is a function of the form
\begin{displaymath}
y = mx + c
\end{displaymath}
```

↓Input

Output:

↑Output

A linear function is a function of the form

$$y = mx + c$$

↓Output

The `equation` environment is the same as the `displaymath` environment, except that the equation is numbered. Substituting `equation` for `displaymath` in the above example:

↑Input

```
A linear function is a function of the form
\begin{equation}
y = mx + c
\end{equation}
```

↓Input

results in the following output:

↑Output

A linear function is a function of the form

$$y = mx + c \tag{9.1}$$

↓Output

Recall from Section 5.5 that we can cross-reference most things that LaTeX automatically numbers using `\ref` and `\label`. Equations can be cross-referenced in the same way:

↑Input

```
Equation~\ref{eqn:linear} is a linear function.
\begin{equation}
\label{eqn:linear}
f(x) = mx + c
\end{equation}
```

↓Input

↑Output

Equation 9.2 is a linear function.

$$f(x) = mx + c \tag{9.2}$$

↓Output

Note: both the `equation` and the `displaymath` environments are only designed for one line of maths. Therefore you must not have any line breaks or paragraph breaks within them. If you want several aligned equations, you need to use another environment, such as `eqnarray` or `align`. This document does not cover these environments, but if you are interested see *The LaTeX Companion* [3] or *A Guide to LaTeX* [2].

9.3 Mathematical Commands

Most of the commands described in this section may only be used in one of the mathematics environments. If you try to use a mathematics command outside a maths environment you will get a "`Missing $ inserted`" error message.

Table 9.1: Maths Font Changing Commands

Command	Example Input	Corresponding Output
\mathrm{*maths*}	xyz	xyz
\mathsf{*maths*}	xyz	xyz
\mathtt{*maths*}	$\mathtt{xyz}$	$\mathtt{xyz}$
\mathit{*maths*}	xyz	xyz
\mathbf{*maths*}	$\mathbf{xyz}$	$\mathbf{xyz}$
\mathcal{*maths*}	$\mathcal{XYZ}$	$\mathcal{XYZ}$

9.3.1 Maths Fonts

Just as we are able to change text fonts using the commands \textrm, \textbf etc, we can also use commands to change the maths font. Basic maths font changing commands are shown in Table 9.1.

The caligraphic fonts are only available for upper-case characters. Note that if you want actual text to appear in a maths environment you need to either use \mbox{*text*}:

↑Input
```
\begin{displaymath}
x > y \mbox{ and } y < z
\end{displaymath}
```
↓Input

↑Output

$$x > y \text{ and } y < z$$

↓Output

or the command \text{*text*} which is defined in the amsmath package:

↑Input
```
\begin{displaymath}
x > y \text{ and } y < z
\end{displaymath}
```
↓Input

↑Output

$$x > y \text{ and } y < z$$

↓Output

Table 9.2 lists additional font commands supplied with the amsmath and amsfonts packages.

9.3.2 Greek Letters

Greek letters that differ from the corresponding Roman letters are obtained by placing a backslash in front of the name. Lower case Greek letters are shown in Table 9.3 and upper case Greek letters are shown in Table 9.4.

Note that there is no omicron as this is the same as a lower case *o*. There are also some variants of certain symbols, such as `\vartheta` as opposed to `\theta`.

9.3.3 Subscripts and Superscripts

Subscripts are obtained either by the command

`\sb{`*maths*`}` Definition

or by the special character:

`_{`*maths*`}` Definition

Superscripts are obtained either by the command

`\sp{`*maths*`}` Definition

or by the special character:

`^{`*maths*`}` Definition

Examples:

1. This example uses `\sb` and `\sp`:

↑Input

```
\begin{displaymath}
y = x\sb{1}\sp{2} + x\sb{2}\sp{2}
\end{displaymath}
```

↓Input

↑Output

$$y = x_1^2 + x_2^2$$

↓Output

2. This example uses `_` and `^`

Table 9.2: The `amsfonts`* and `amsmath`† Font Commands

Command	Example Input	Example Output
*`\mathbb{`*maths*`}`	`$\mathbb{A+B=C}$`	$\mathbb{A+B=C}$
*`\mathfrak{`*maths*`}`	`$\mathfrak{A+B=C}$`	$\mathfrak{A+B=C}$
†`\boldsymbol{`*maths*`}`	`$\boldsymbol{A+B=C}$`	$\boldsymbol{A+B=C}$
†`\pmb{`*symbol*`}`	`$\pmb{+-=}$`	$\pmb{+-=}$

Table 9.3: Lower Case Greek Letters

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

Table 9.4: Upper Case Greek Letters

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

↑Input

```
\begin{displaymath}
y = x_{1}^{2} + x_{2}^{2}
\end{displaymath}
```

↓Input

↑Output

$$y = x_1^2 + x_2^2$$

↓Output

3. Recall that mandatory arguments only consisting of one character don't need to be grouped, so the above code can also be written as:

↑Input

```
\begin{displaymath}
y = x_1^2 + x_2^2
\end{displaymath}
```

↓Input

↑Output

$$y = x_1^2 + x_2^2$$

↓Output

This is simpler than the above two examples.

4. Subscripts and superscripts can also be nested:

↑Input

```
\begin{displaymath}
f(x) = e^{x_1}
\end{displaymath}
```

↓Input

↑Output

$$f(x) = e^{x_1}$$

↓Output

This example is slightly incorrect as e isn't actually a variable and shouldn't be typeset in italic. The correct way to do this is:

↑Input

```
\begin{displaymath}
f(x) = \mathrm{e}^{x_1}
\end{displaymath}
```

↓Input

↑Output

$$f(x) = \mathrm{e}^{x_1}$$

↓Output

If you are going to use e a lot, it will be simpler to define a new command to do this:

↑Input

```
\newcommand{\e}{\mathrm{e}}

\begin{displaymath}
f(x_1, x_2) = \e^{x_1^2} + \e^{x_2^2}
\end{displaymath}
```

↓Input

↑Output

$$f(x_1, x_2) = \mathrm{e}^{x_1^2} + \mathrm{e}^{x_2^2}$$

↓Output

9.3.4 Functional Names

Functions such as log and tan can't simply be typed in as `log` or `tan` otherwise they will come out looking like the variables l times o times g (log) or t times a times n (tan). Instead you should use one of the commands listed in Table 9.5.

Table 9.5: Function Names

`\arccos`	`\arcsin`	`\arctan`	`\arg`	`\cos`	`\cosh`
`\cot`	`\coth`	`\csc`	`\deg`	`\det`	`\dim`
`\exp`	`\gcd`	`\hom`	`\inf`	`\ker`	`\lg`
`\lim`	`\liminf`	`\limsup`	`\ln`	`\log`	`\max`
`\min`	`\Pr`	`\sec`	`\sin`	`\sinh`	`\sup`
`\tan`	`\tanh`				

Of these functions, the following functions can have limits by using the subscript command `_` or the superscript command `^`:

`\det`	`\gcd`	`\inf`	`\lim`	`\liminf`
`\limsup`	`\max`	`\min`	`\Pr`	`\sup`

Examples:

1. This example uses the cos and sin functions and also the Greek letter theta.

↑Input

```
\begin{displaymath}
z = r(\cos\theta + i\sin\theta)
\end{displaymath}
```

↓Input

↑Output

$$z = r(\cos\theta + i\sin\theta)$$

↓Output

2. This example has a limit. The command `\infty` is the infinity symbol ∞, and the command `\to` displays an arrow pointing to the right. Note the use of `_`

```
\begin{displaymath}
\lim_{x\to\infty} f(x)
\end{displaymath}
```

↑Input ↓Input

↑Output

$$\lim_{x\to\infty} f(x)$$

↓Output

3. This is another example of a functional name using a subscript:

```
\begin{displaymath}
\min_x f(x)
\end{displaymath}
```

↑Input ↓Input

↑Output

$$\min_x f(x)$$

↓Output

In addition, the following commands are also available:

Command	Example Input	Example Output
`\bmod`	`$m \bmod n$`	$m \bmod n$
`\pmod{`*maths*`}`	`$m \pmod{n}$`	$m \pmod{n}$

If you want a function that isn't specified in Table 9.5, you can use the command

`\operatorname{`*operator name*`}` Definition

or

`\operatornamewithlimits{`*operator name*`}` Definition

both of which are defined in the `amsmath` package.

The second of these commands, `\operatornamewithlimits`, allows you to have a function that can take limits using the `_` or `^` commands, just like `\lim`, `\min` etc.

Examples

1. Suppose we want a function called card, which represents the cardinality of a set $\mathcal{S}$:

```
\begin{displaymath}
n = \operatorname{card}(\mathcal{S})
\end{displaymath}
```

↑Input ↓Input

↑Output

$$n = \operatorname{card}(\mathcal{S})$$

↓Output

In this example `\mathcal` is used as sets are usually represented in a caligraphic font.

2. It's a bit cumbersome having to keep typing `\operatorname{card}` everytime you want card, a better thing to do would be to define a new command called `\card`.

```
\newcommand{\card}{\operatorname{card}}

\begin{displaymath}
n = \card(\mathcal{S})
\end{displaymath}
```

↑Input ↓Input

↑Output

$$n = \operatorname{card}(\mathcal{S})$$

↓Output

3. Let's have an example of an operator that takes a limit:

```
\newcommand{\mode}{\operatornamewithlimits{mode}}

\begin{displaymath}
x_m = \mode_{x \in \mathcal{S}}(x)
\end{displaymath}
```

↑Input

↓Input

↑Output

$$x_m = \operatorname*{mode}_{x \in \mathcal{S}}(x)$$

↓Output

9.3.5 Fractions

Fractions are created using the command

`\frac{`*numerator*`}{`*denominator*`}` Definition

Examples:

1. A simple fraction:

↑Input

```
\begin{displaymath}
\frac{1}{1+x}
\end{displaymath}
```

↓Input

↑Output

$$\frac{1}{1+x}$$

↓Output

2. A nested fraction:

↑Input

```
\begin{displaymath}
\frac{1+\frac{1}{x}}{1+x+x^2}
\end{displaymath}
```

↓Input

↑Output

$$\frac{1+\frac{1}{x}}{1+x+x^2}$$

↓Output

3. A derivative:

```
\begin{displaymath}
f'(x) = \frac{df}{dx}
\end{displaymath}
```

↑Input ↓Input

↑Output

$$f'(x) = \frac{df}{dx}$$

↓Output

Again, as with e, the differential operator 'd' should be in an upright font as it is not a variable:

```
\begin{displaymath}
f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}
\end{displaymath}
```

↑Input ↓Input

↑Output

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}$$

↓Output

4. The above example is rather cumbersome, particularly if you have a lot of derivatives, so it might be easier to define a new command:

```
\newcommand{\deriv}[2]{\frac{\mathrm{d}#1}{\mathrm{d}#2}}

\begin{displaymath}
f'(x) = \deriv{f}{x}
\end{displaymath}
```

↑Input ↓Input

↑Output

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}$$

↓Output

5. Partial derivatives can be obtained similarly using the command `\partial` to display the partial derivative symbol:

↑Input

```
\newcommand{\pderiv}[2]{\frac{\partial #1}{\partial #2}}

\begin{displaymath}
f_x = \pderiv{f}{x}
\end{displaymath}
```

↓Input

↑Output

$$f_x = \frac{\partial f}{\partial x}$$

↓Output

6. A double partial derivative:

↑Input

```
\begin{displaymath}
f_{xy} = \frac{\partial^2 f}{\partial x \partial y}
\end{displaymath}
```

↓Input

↑Output

$$f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$$

↓Output

9.3.6 Roots

Roots are obtained using the command

`\sqrt[`*order*`]{`*maths*`}` Definition

without the optional argument *order* it will produce a simple square root. Cubic roots etc can be obtained using the optional argument.

Examples:

1. A square root:

↑Input

```
\begin{displaymath}
\sqrt{a+b}
\end{displaymath}
```

↓Input

↑Output

$$\sqrt{a+b}$$

↓Output

2. A cubic root:

↑Input

```
\begin{displaymath}
\sqrt[3]{a+b}
\end{displaymath}
```

↓Input

↑Output

$$\sqrt[3]{a+b}$$

↓Output

3. An nth root:

↑Input

```
\begin{displaymath}
\sqrt[n]{a+b}
\end{displaymath}
```

↓Input

↑Output

$$\sqrt[n]{a+b}$$

↓Output

Table 9.6: Relational Symbols

\approx	$\approx$	\asymp	$\asymp$	\bowtie	$\bowtie$
\cong	$\cong$	\dashv	$\dashv$	\doteq	$\doteq$
\equiv	$\equiv$	\frown	$\frown$	\ge or \geq	$\geq$
\gg	$\gg$	\in	$\in$	\le or \leq	$\leq$
\ll	$\ll$	\mid or |	$\mid$	\models	$\models$
\neq	$\neq$	\ni	$\ni$	\notin	$\notin$
\parallel	$\parallel$	\prec	$\prec$	\preceq	$\preceq$
\perp	$\perp$	\propto	$\propto$	\sim	$\sim$
\simeq	$\simeq$	\smile	$\smile$	\sqsubseteq	$\sqsubseteq$
\sqsupseteq	$\sqsupseteq$	\subset	$\subset$	\subseteq	$\subseteq$
\succ	$\succ$	\succeq	$\succeq$	\supset	$\supset$
\supseteq	$\supseteq$	\vdash	$\vdash$		

9.3.7 Mathematical Symbols

Relational symbols are shown in Table 9.6. If you want a negation that is not shown, you can obtain it by preceeding the symbol with the command `\not`. For example: `\not\subset` produces the symbol $\not\subset$.

Binary operator signals are shown in Table 9.7, and arrow symbols are shown in Table 9.8.

Symbols that can have limits are shown in Table 9.9. The size of these symbols depends on whether they are in displayed maths or in-line maths. Examples:

1. Displayed Maths

↑Input

```
\begin{displaymath}
f(x) = \sum_{i=1}^{n} x_i + \prod_{i=1}^{n} x_i
\end{displaymath}
```

↓Input

↑Output

$$f(x) = \sum_{i=1}^{n} x_i + \prod_{i=1}^{n} x_i$$

↓Output

2. In-line Maths

↑Input

```
\begin{math}
f(x) = \sum_{i=1}^{n} x_i + \prod_{i=1}^{n} x_i
\end{math}
```

Table 9.7: Binary Operator Symbols

\amalg	∐	\ast	∗	\bullet	•
\bigcirc	◯	\bigtriangledown	▽	\bigtriangleup	△
\cap	∩	\cdot	·	\circ	∘
\cup	∪	\dagger	†	\ddagger	‡
\diamond	⋄	\div	÷	\mp	∓
\odot	⊙	\ominus	⊖	\oplus	⊕
\oslash	⊘	\otimes	⊗	\pm	±
\setminus	\	\sqcap	⊓	\sqcup	⊔
\star	⋆	\times	×	\triangleleft	◁
\triangleright	▷	\uplus	⊎	\vee	∨
\wedge	∧	\wr	≀		

Table 9.8: Arrow Symbols

\downarrow	↓	\Downarrow	⇓
\hookleftarrow	↩	\hookrightarrow	↪
\leftarrow or \gets	←	\Leftarrow	⇐
\leftharpoondown	↽	\leftharpoonup	↼
\leftrightarrow	↔	\Leftrightarrow	⇔
\longleftarrow	⟵	\Longleftarrow	⟸
\longleftrightarrow	⟷	\Longleftrightarrow	⟺
\longmapsto	⟼	\longrightarrow	⟶
\Longrightarrow	⟹	\mapsto	↦
\nearrow	↗	\nwarrow	↖
\rightarrow or \to	→	\Rightarrow	⇒
\rightharpoondown	⇁	\rightharpoonup	⇀
\rightleftharpoons	⇌	\searrow	↘
\swarrow	↙	\uparrow	↑
\Uparrow	⇑	\updownarrow	↕
\Updownarrow	⇕		

Table 9.9: Symbols with Limits

\sum	∑	\int	∫	\oint	∮
\prod	∏	\coprod	∐	\bigcap	⋂
\bigcup	⋃	\bigsqcup	⨆	\bigvee	⋁
\bigwedge	⋀	\bigodot	⨀	\bigotimes	⨂
\bigoplus	⨁	\biguplus	⨄		

↓Input

↑Output

$$f(x) = \sum_{i=1}^{n} x_i + \prod_{i=1}^{n} x_i$$

↓Output

Ellipsis commands are shown in Table 9.10.

Table 9.10: Ellipses

`\ldots`	$\ldots$	`\cdots`	$\cdots$
`\vdots`	$\vdots$	`\ddots`	$\ddots$

Examples:

1. Low ellipsis: This example uses the command `\forall` to produce the 'for all' symbol ∀, and it also uses \␣ (backslash space) to make a space before the for all symbol:

↑Input

```
\begin{displaymath}
a_ix_i = b_i \ \forall i = 1,\ldots, n
\end{displaymath}
```

↓Input

↑Output

$$a_i x_i = b_i \ \forall i = 1, \ldots, n$$

↓Output

2. Centred ellipsis:

↑Input

```
\begin{displaymath}
y = a_1 + a_2 + \cdots + a_n
\end{displaymath}
```

↓Input

↑Output

$$y = a_1 + a_2 + \cdots + a_n$$

↓Output

For other symbol commands, see *A Guide to LaTeX* [2] or *The LaTeX Companion* [3].

Exercise 22 (Maths: Fractions and Symbols)

This exercise uses a fraction, a square root, subscripts, superscripts and symbols. Try reproducing the following output:

↑Output

The quadratic equation

$$\sum_{i=0}^{2} a_i x^i = 0$$

has solutions given by

$$x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2a_0}}{2a_2}$$

↓Output

Again you can download or view the solution.

9.3.8 Delimiters

Placing brackets around a tall object in maths mode, such as fractions, does not look right if you use normal sized brackets. For example:

↑Input

```
\begin{displaymath}
(\frac{1}{1+x})
\end{displaymath}
```

↓Input

↑Output

$$(\frac{1}{1+x})$$

↓Output

Under such circumstances, it is better to use the commands:

`\left`*delimiter* Definition

and

`\right`*delimiter* Definition

Note that you must always have matching `\left` and `\right` commands, although the delimiters used may be different. Available delimiters are shown in Table 9.11.

If you want one of the delimiters to be invisible, use a `.` (full stop) as the delimiter.
Examples:

1. Round bracket delimiters:

Table 9.11: Delimiters

`(`	(	`)`	)	`[`	[	`]`	]
`\{`	{	`\}`	}	`|`	|	`\\|`	‖
`/`	/	`\backslash`	\	`\langle`	⟨	`\rangle`	⟩
`\lfloor`	⌊	`\rfloor`	⌋	`\lceil`	⌈	`\rceil`	⌉
`\uparrow`	↑	`\downarrow`	↓	`\Uparrow`	⇑	`\Downarrow`	⇓
`\updownarrow`	↕	`\Updownarrow`	⇕				

```
\begin{displaymath}
\left(
\frac{1}{1+x}
\right)
\end{displaymath}
```

↑Input ↓Input

↑Output

$$\left(\frac{1}{1+x}\right)$$

↓Output

2. Vertical bar delimiters:

```
\begin{displaymath}
\left|
\frac{1}{1+x}
\right|
\end{displaymath}
```

↑Input ↓Input

↑Output

$$\left|\frac{1}{1+x}\right|$$

↓Output

3. Delimiters don't have to match:

↑Input

```
\begin{displaymath}
\left[\frac{1}{1+x}\right\rangle
\end{displaymath}
```

↓Input

↑Output

$$\left[\frac{1}{1+x}\right\rangle$$

↓Output

We have now learnt enough to reproduce the equation shown in Chapter 1:

↑Input

```
\newcommand{\pderiv}[2]{\frac{\partial #1}{\partial #2}}
\newcommand{\e}{\mathrm{e}}

\begin{displaymath}
\pderiv{^2\mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
\left(
  \pderiv{v_i}{\rho_i} \frac{\e^{v_i}}{1-\e^{v_i}}
  + v_i \frac{\e^{v_i}\pderiv{v_i}{\rho_i}(1-\e^{v_i})
              +\e^{2v_i}\pderiv{v_i}{\rho_i}}{(1-\e^{v_i})^2}
\right)
\end{displaymath}
```

↓Input

↑Output

$$\frac{\partial^2 \mathcal{L}}{\partial {z_i^\rho}^2} = -\frac{\partial \rho_i}{\partial z_i^\rho}\left(\frac{\partial v_i}{\partial \rho_i}\frac{\mathrm{e}^{v_i}}{1-\mathrm{e}^{v_i}} + v_i\frac{\mathrm{e}^{v_i}\frac{\partial v_i}{\partial \rho_i}(1-\mathrm{e}^{v_i}) + \mathrm{e}^{2v_i}\frac{\partial v_i}{\partial \rho_i}}{(1-\mathrm{e}^{v_i})^2}\right)$$

↓Output

Note: The above code looks a bit complicated, and there are so many braces that it can be easy to lose track, so here are some ways of making it a little easier to type:

1. Whenever you start a new environment type in the `\begin` and `\end` bits first, and then insert whatever goes inside the environment. This ensures that you always have a matching `\begin` and `\end`.

2. Whenever you type any braces, always type the opening and closing braces first, and then insert whatever goes in between. This will ensure that your braces always match up.

So keeping these notes in mind, let's try typing in the code in a methodical manner:

1. Start the `displaymath` environment:

```
\begin{displaymath}
\end{displaymath}
```

2. We now need a partial derivative:

```
\begin{displaymath}
\pderiv{}{}
\end{displaymath}
```

3. Let's do the first argument. This partial derivative is actually a double derivative, which means we need a squared bit on the top along with a caligraphic L:

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{}
\end{displaymath}
```

4. The second argument is the z_i^ρ squared bit. This is a nested superscript `{z_i^\rho}^2`:

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{{z_i^\rho}^2}
\end{displaymath}
```

5. We can do the next partial derivative in the same way. This one is slightly easier to do:

```
\begin{displaymath} \pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
\end{displaymath}
```

6. Delimiters also need to occur in pairs, like curly braces and `\begin` and `\end`, so let's do them next:

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
```

```
  \left(
  \right)
\end{displaymath}
```

↓Input

7. Now we need to do the bits inside the brackets. First of all we have yet another partial derivative:

↑Input

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
  \left(
    \pderiv{v_i}{\rho_i}
  \right)
\end{displaymath}
```

↓Input

8. Now we have a fraction:

↑Input

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
  \left(
    \pderiv{v_i}{\rho_i} \frac{\e^{v_i}}{1-\e^{v_i}}
  \right)
\end{displaymath}
```

↓Input

9. This is followed by v_i times another fraction:

↑Input

```
\begin{displaymath}
\pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
  \left(
    \pderiv{v_i}{\rho_i} \frac{\e^{v_i}}{1-\e^{v_i}}
    + v_i \frac{}{}
  \right)
\end{displaymath}
```

↓Input

10. This fraction is quite complicated. The bottom part of the fraction is easier than the top, so let's do that first:

↑Input

```
\begin{displaymath} \pderiv{^2 \mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
  \left(
    \pderiv{v_i}{\rho_i} \frac{\e^{v_i}}{1-\e^{v_i}}
    + v_i \frac{}{(1-\e^{v_i})^2}
```

```
  \right)
\end{displaymath}
```

↓Input

11. Now it's time for the top part of the fraction. It's a bit complicated, so let's break it down:

    (a) The first term is:

    ```
    \e^{v_i}
    ```

    (b) The next term is another partial derivative:

    ```
    \pderiv{v_i}{\rho_i}
    ```

    (c) Then we have:

    ```
    (1-\e^{v_i})
    ```

    (d) Next we have to add on:

    ```
    +\e^{2v_i}
    ```

    (e) And finally we have the last term:

    ```
    \pderiv{v_i}{\rho_i}
    ```

12. Putting it all together, we have:

↑Input

```
\begin{displaymath}
\pderiv{^2\mathcal{L}}{{z_i^\rho}^2} =
-\pderiv{\rho_i}{z_i^\rho}
\left(
  \pderiv{v_i}{\rho_i} \frac{\e^{v_i}}{1-\e^{v_i}}
  + v_i \frac{\e^{v_i}\pderiv{v_i}{\rho_i}(1-\e^{v_i})
              +\e^{2v_i}\pderiv{v_i}{\rho_i}}{(1-\e^{v_i})^2}
\right)
\end{displaymath}
```

↓Input

13. And remember that if you haven't already defined `\pderiv` and `\e`, you will need to do that in the preamble

↑Input

```
\newcommand{\pderiv}[2]{\frac{\partial #1}{\partial #2}}
\newcommand{\e}{\mathrm{e}}
```

↓Input

(Note that if we hadn't defined these two commands, the code would have had to have been far more complicated.)

9.3.9 Arrays

Mathematical structures such as matrices and vectors require elements to be arranged in rows and columns. Just as we can align material in rows and columns in text mode using the `tabular` environment, we can do the same in maths mode using the `array` environment. The `array` environment has the same format as the `tabular` environment, however it must be in maths mode. Examples:

1. A simple array, all three columns are right justified:

↑Input

```
\begin{displaymath}
\begin{array}{rrr}
0 & 1 & 19\\
-6 & 10 & 200
\end{array}
\end{displaymath}
```

↓Input

↑Output

$$\begin{array}{rrr} 0 & 1 & 19 \\ -6 & 10 & 200 \end{array}$$

↓Output

2. Let's add some delimiters:

↑Input

```
\begin{displaymath}
\left(
\begin{array}{rrr}
0 & 1 & 19\\
-6 & 10 & 200
\end{array}
\right)
\end{displaymath}
```

↓Input

↑Output

$$\left(\begin{array}{rrr} 0 & 1 & 19 \\ -6 & 10 & 200 \end{array} \right)$$

↓Output

3. This example uses an invisible delimiter:

↑Input

```
\begin{displaymath}
f(x) =
\left\{
\begin{array}{rl}
-1 & x < 0\\
0 & x = 0\\
+1 & x > 0
\end{array}
\right.
\end{displaymath}
```

↓Input

↑Output

$$f(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ +1 & x > 0 \end{cases}$$

↓Output

9.3.10 Vectors

Vectors can be produced using the command:

`\vec{`*variable*`}` Definition

Example:

↑Input

```
\begin{displaymath}
\vec{x}
\end{displaymath}
```

↓Input

↑Output

$$\vec{x}$$

↓Output

These days it is customary to typeset vectors in bold. This can be done by redefining the `\vec` command. You could use `\mathbf`, for example:

↑Input
```
\renewcommand{\vec}[1]{\mathbf{#1}}
\begin{displaymath}
\vec{x}\cdot\vec{\xi}
\end{displaymath}
```
↓Input

↑Output

$$\mathbf{x}\cdot\xi$$

↓Output

however, as you can see, the Greek letter ξ has not come out in bold. Here's an alternative (using `\boldsymbol` defined in the `amsfonts` package):

↑Input
```
\renewcommand{\vec}[1]{\boldsymbol{#1}}
\begin{displaymath}
\vec{x}\cdot\vec{\xi} = z
\end{displaymath}
```
↓Input

↑Output

$$\boldsymbol{x}\cdot\boldsymbol{\xi} = z$$

↓Output

Exercise 23 (Maths: Vectors and Arrays)

Try to produce the following:

↑Output

$$\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \end{pmatrix} = \boldsymbol{y}$$

↓Output

As before, you can download or view the solution.

9.3.11 Mathematical Spacing

LaTeX deals with mathematical spacing fairly well, but sometimes you may find you want to adjust the spacing yourself. Available spacing commands are listed in Table 9.12.

Exercise 24 (More Mathematics)

This exercise uses the spacing command `\qquad`. It also has a function name, diag, and it uses the `\forall` and ellipses symbols. It also redefines the `\vec`

Table 9.12: Mathematical Spacing Commands

Command	Example Input	Example Output
	`$AB$`	AB
`\thinspace` or `\,`	`$A\,B$`	$A\,B$
`\medspace` or `\:`	`$A\:B$`	$A\:B$
`\thickspace` or `\;`	`$A\;B$`	$A\;B$
`\quad`	`$A\quad B$`	$A \quad B$
`\qquad`	`$A\qquad B$`	$A \qquad B$
`\negthinspace` or `\!`	`$A\!B$`	$A\!B$
`\negmedspace`	`$A\negmedspace B$`	$A\negmedspace B$
`\negthickspace`	`$A\negthickspace B$`	$A\negthickspace B$

command, as was done in the previous section, and it uses delimiters and the `array` environment, as well as using subscripts and superscripts.

Try to reproduce the following output:

↑Output

The set of linear equations:

$$a_i x_i = b_i \qquad \forall i = 1, \ldots, n$$

can be written as a matrix equation:

$$\text{diag}(\boldsymbol{a})\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^T$, $\boldsymbol{b} = (b_1, \ldots, b_n)^T$ and

$$\text{diag}(\boldsymbol{a}) = \begin{bmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_n \end{bmatrix}$$

↓Output

Again, you can download or view the solution.

Chapter 10

Defining Environments

Just as you can define new commands, you can also define new environments. The command

`\newenvironment{`*env-name*`}[`*n-args*`][`*default*`]{`*begin-code*`}{`*end-code*`}` Definition

is used to define a new environment. As with new commands, you can use the optional argument *n-args* to define an environment with arguments, and *default* to define an environment with an optional argument.

The first argument *env-name* is the name of your new environment. Remember that the environment name must not have a backslash. The mandatory arguments *begin-code* and *end-code* indicate what LaTeX should do at the beginning and end of the environment. Let's first consider an example of an environment without any arguments. Let's make an environment called, say, `exercise` that prints **Exercise** in bold and typesets the contents of the environment in italic. In other words, we want the following code:

↑Input
```
\begin{exercise}
This is a sample.
\end{exercise}
```
↓Input

to produce the following output:

↑Output

Exercise
This is a sample.

↓Output

Let's first consider what we want this environment to do: we can get the word "Exercise" in bold by simply doing `\textbf{Exercise}`, and the italic font can be obtained by using the `itshape` environment. So, at the start of our new environment we need to do `\textbf{Exercise}` and we need to begin the `itshape` environment, and at the end of our new environment we need to end the `itshape` environment:

↑Input
```
\newenvironment{exercise}%          environment name
{\textbf{Exercise}\begin{itshape}}% begin code
{\end{itshape}}%                    end code
```

↓Input

Let's try it out:

↑Input

```
\begin{exercise}
This is a sample.
\end{exercise}
```

↓Input

Exercise *This is a sample.*

Output

Not quite right. Let's put a paragraph break after **Exercise**, and put one before it as well. The command `\par` can be used to make a paragraph break:

↑Input

```
\newenvironment{exercise}%                    environment name
{\par\textbf{Exercise}\begin{itshape}\par}% begin code
{\end{itshape}}%                              end code
```

↓Input

Let's have a look at the output now:

↑Output

Exercise
This is a sample.

↓Output

One more thing, we need to remove the paragraph indentation. This can be done using the command `\noindent`:

↑Input

```
\newenvironment{exercise}
{\par\noindent\textbf{Exercise}\begin{itshape}\par\noindent}
{\end{itshape}}
```

↓Input

Now let's modify our code so that the environment takes an argument. The argument should indicate the exercise topic. For example, the following code:

↑Input

```
\begin{exercise}{An Example}
This is a sample.
\end{exercise}
```

↓Input

should produce the following result:

↑Output

Exercise (An Example)
This is a sample.

↓Output

As with `\newcommand`, `#1` is used to indicate the first argument. We can now modify the code as follows (modifications are indicated `like this`):

↑Input

```
\newenvironment{exercise}[1]%
{\par\noindent\textbf{Exercise (#1)}\begin{itshape}\par\noindent}%
{\end{itshape}}
```

↓Input

Exercise 25 (Defining a New Environment)

If you did any of the exercises from Exercise 8 to Exercise 18, go back to the document you created and define the `exercise` environment as in the example above. Then try creating some exercises using this environment. You could, maybe, put an exercise in the first chapter, and then another one in the second chapter.

Then try modifying the environment so that it puts a bit of vertical space before and after the environment using `\vspace{`*length*`}`. Again you can download or view an example.

Chapter 11

Counters

As we have seen, LaTeX automatically generates numbers for chapters, sections, equations etc. These numbers are stored in counters. The names of these counters is usually the same as the name of the object with which it is associatated but without any backslash. For example, the `\chapter` command has an associated counter called `chapter`, the `\footnote` command has an associated counter called `footnote`, the `equation` environment has an associated counter called `equation`, the `figure` environment has an associated counter called `figure` and the `table` environment has an associated counter called `table`. There is also a counter called `page` that keeps track of the current page number.

The value of a counter can be displayed using the command

`\the`*counter* Definition

where *counter* is the name of the associated counter. Note that *counter* does not go in curly braces and adjoins `\the` (e.g. `\thepage`, `\thesection`, `\thechapter`). Example:

```
This page is Page~\thepage.
The current chapter is Chapter~\thechapter.
```
↑Input ↓Input

This page is Page 115. The current chapter is Chapter 11. Output

New counters can be created using the command:

`\newcounter{`*counter-name*`}[`*outer-counter*`]` Definition

The mandatory argument *counter-name* is the name of your new counter (no backslash in the name). For example, let's define a counter called `exercise` to keep track of each exercise.

```
\newcounter{exercise}
```
Input

we can now display the value of the counter using the command `\theexercise`. At the moment the counter has the value zero, the value can be changed using one of the following commands:

`\stepcounter{`*counter*`}`	Increments *counter* by 1
`\refstepcounter{`*counter*`}`	As above, but allows you to cross-reference the counter using `\label` and `\ref`
`\setcounter{`*counter*`}{`*num*`}`	Sets the counter to *num*
`\addtocounter{`*counter*`}{`*num*`}`	Adds *num* to *counter*

A couple of the commands above take a number *num* as one of the arguments. If you want to use another counter for this argument, you need to use

```
\value{counter}
```
Definition

For example, if you want to set our new `exercise` counter to the same value as the `page` counter, you would do

```
\setcounter{exercise}{\value{page}}
```
Input

Let's go back to the `exercise` environment you created in Exercise 25. The exercises really ought to have an associated number, and this number should be incremented each time we use the exercise environment. So let's modify our code to do this. Modifications are illustrated like this:

↑Input
```
\newcounter{exercise}

\newenvironment{exercise}[1]%
{\refstepcounter{exercise}\vspace{10pt}\par\noindent
\textbf{Exercise \theexercise\ (#1)}
\begin{itshape}\par\noindent\vspace{10pt}}%
{\end{itshape}\vspace{10pt}\par}
```
↓Input

Note that the counter needs to be incremented before it is used. Since we've used `\refstepcounter` instead of `\stepcounter` we can cross-reference our `exercise` environment:

↑Input
```
Exercise~\ref{ex:simple} is a simple exercise.

\begin{exercise}{Simple Exercise}
\label{ex:simple}
This is a simple exercise.
\end{exercise}
```
↓Input

↑Output

Exercise 1 is a simple exercise.

Exercise 1 (Simple Exercise)
This is a simple exercise.

↓Output

The counter representation can be changed by redefining `\theexercise` using the command `\renewcommand`. The following commands can be used to display the counter:

`\arabic{`*counter*`}`	Arabic number (1, 2, 3, ...)
`\Roman{`*counter*`}`	Uppercase Roman numeral (I, II, III, ...)
`\roman{`*counter*`}`	Lowercase Roman numeral (i, ii, iii, ...)
`\alph{`*counter*`}`	Lowercase letter (a, b, c, ..., z)
`\Alph{`*counter*`}`	Uppercase letter (A, B, C, ..., Z)
`\fnsymbol{`*counter*`}`	A footnote symbol (∗ † ‡ § ¶ ‖ ∗∗ †† ‡‡)

For example, to make the chapter numbers appear as uppercase Roman numerals you would do:

```
\renewcommand{\thechapter}{\Roman{chapter}}
```
Input

You may have noticed that `\newcounter` has an optional argument *outer-counter*. This is for use if you require the new counter to be reset every time *outer-counter* is incremented. For example, the section numbers in the `report` class are dependent on the chapter numbers. Each time a new chapter is started, the section numbers are reset. Suppose we want our `exercise` counter to be dependent on the `chapter` counter, we would do

```
\newcounter{exercise}[chapter]
```
Input

We now need to modify `\theexercise` so that it includes the chapter number:

```
\renewcommand{\theexercise}{\thechapter.\arabic{exercise}}
```
Input

Notice the use of `\thechapter` instead of, say, `\arabic{chapter}`. By using `\thechapter` we don't need to keep track of the `chapter` counter format.

Exercise 26 (Using Counters)

Modify the document from Exercise 25 so that the `exercise` environment has a counter. Make the counter dependent on the chapter. You can download or view an example.

Chapter 12

Lengths

Lengths are commands that store dimensions (such as 1in, 5cm, 8.25mm.) These are used to determine page layouts etc. For example, the page width is given by the length `\pagewidth`, and the height of the main body of text is given by `\textheight`. The paragraph indentation is given by `\parindent` and the gap between paragraphs is given by `\parskip`. Acceptable units of measurement are shown in Table 12.1.

Example: The default paragraph indentation `\parindent` is usually around 15pt, but we can change this if we like. To change a length you need to use the command:

`\setlength{`*cmd*`}{`*length*`}` Definition

where *cmd* is the particular length command (e.g. `\parindent`) and *length* is the new length. To display the value of a length do:

`\the`*cmd* Definition

Example:

Input

```
\setlength{\parindent}{0pt}

This is the first paragraph.
```

Table 12.1: Units of Measurement

`pt`	point: 72.27pt = 1in
`in`	inch: 1in = 25.4mm
`mm`	millimetre: 1mm=2.845pt
`cm`	centimetre: 1cm = 10mm
`ex`	height of the letter x in the current font
`em`	width of the letter M in the current font
`sp`	scaled point: 1sp = 65536pt
`bp`	big point (or PostScript point): 72bp = 1in
`dd`	didôt point: 1dd=0.376mm
`pc`	pica: 1pc=12pt
`cc`	cicero: 1cc=12dd
`mu`	math unit: 18mu = 1em

```
This is the second paragraph.
The paragraph indentation is \the\parindent.
```

↓Input

↑Output

This is the first paragraph.
This is the second paragraph. The paragraph indentation is 0.0pt.

↓Output

A rubber length is a length that has a certain amount of elasticity. This enables you to specify your desired length with a certain amount of flexibility that will allow LaTeX to stretch or contract the text to get the body of text as flushed with the margins as possible.

For example, the paragraph gap `\parskip` is usually set to `0pt plus 1pt`. This means that the prefered gap is 0pt but LaTeX can stretch it up to 1pt to help prevent the page from having a ragged bottom. Let's further modify the above example:

↑Input

```
\setlength{\parindent}{0pt}
\setlength{\parskip}{10pt plus 1pt minus 1pt}

This is the first paragraph.

This is the second paragraph.
The paragraph indentation is \the\parindent.
```

↓Input

↑Output

This is the first paragraph.

This is the second paragraph. The paragraph indentation is 0.0pt.

↓Output

In this example, the prefered paragraph gap is 10pt but it will allow for a deviation of up to plus or minus 1pt.

If you want to change any of the page layout lengths (such as `\textwidth`), the easiest way to do it is to use the `geometry` package. This package should have been installed when you installed `MiKTeX`, and the documentation should be in one of the subdirectories of `\texmf\doc\latex`. Using an example from the `geometry` documentation: suppose you want the total text area to be 6.5in wide and 8.75in high, with a left margin 0.4in from the left edge, no header, and the first line of the page to be 1.2in from the top of the paper, then you would do:

```
\usepackage[body={6.5in,8.75in},top=1.2in,left=0.4in,nohead]{geometry}
```

Chapter 13

Common Errors

- If the only message that gets printed to the screen is:

  ```
  Bad command or file name
  ```

  then you have either mistyped the command name, or you don't have LaTeX installed on your computer, or your path hasn't been set up correctly. First check that you have typed the command correctly, then check to see if you have MiKTeX installed. Failing that, contact your system adminstrator for help.

- If you get the message (or something similar):

  ```
  This is TeX, Version 3.14159 (Web2C 7.3.1)
  ! I can't find file `sample'.
  <*> sample

  Please type another input file name:
  ```

  then you have either misspelt the filename or you are in the wrong directory. If you have misspelt the filename, simply type in the correct name at the prompt. If you are in the wrong directory or you want to quit, type `X` followed by the return character ↵. This is an error that you may encounter if you are using notepad and the MS-DOS Prompt, as typing errors may occur, or you may forget to change to the correct directory. To check you are in the right directory, you can type:

  ```
  dir
  ```

  at the MS-DOS prompt. This will list the contents of the directory. If you are certain that you have spelt the filename correctly and that you are in the right directory, there may be something wrong with your path, in which case contact your system administrator. You are unlikely to get this error with WinEdt or TeXnicCenter as they set the directory, and you only need to click a button, so you won't get any typing errors.

- Error messages will usually look something like:

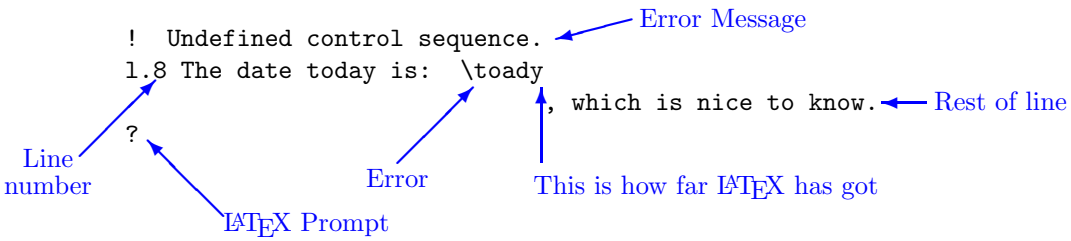
  ```
  ! Undefined control sequence.
  l.1 \docmentclass
                  [a4paper,11pt]{article}
  ?
  ```

The first line is the error message. In this example I have misspelt the command `\documentclass`. The next line begins with `l.` followed by a number. This is the line number in the source code where the error occurred. In this case the error occurred on line 1. Following the line number is the line LaTeX has processed so far, and staggered on the following line is the remainder of the input line.

Here's another example. Suppose line 8 of my source code looks like:

```
The date today is: \toady, which is nice to know.
```

The error in this case is the misspelling of the command `\today`. The error message will appear as follows:



At the LaTeX prompt, you can either type `h` for a help message, or type `x` to exit LaTeX and go back to your source code and fix the problem.

There follows below a list of common error messages.

13.1 * (No message, just an asterisk prompt!)

You've gone into TeX! This is probably because you've forgotten the `\end{document}`. The asterisk is the TeX prompt. At this point the best thing to do is type `\end{document}` at the prompt (followed by the return character ↵) and hope for the best.

13.2 Argument of \cline has an extra }

If this error occurred on the first line in your `tabular` environment, you may have forgotten the argument to the `tabular` environment.

13.3 Argument of \multicolumn has an extra }

If this error occurred on the first line in your `tabular` environment, you may have forgotten the argument to the `tabular` environment.

13.4 \begin{...} ended by \end{...}

The beginning of your environment doesn't have a matching end.

- Check to make sure you have spelt the name of the environment correctly.

  You will get this error message if you do, say,

  `\end{docment}` (**incorrect**)

instead of

`\end{document}` (**correct**)

- Check that for every `\begin` you have a corresponding `\end` with the same name.

13.5 Bad math environment delimiter

Only a certain type of character may be used as a delimiter (e.g. `( ) [ ] \{ \} | \| .`), check which one you have specified. This error may also occur if you have forgotten a `\right` (Remember to use a `.` if you want an invisible delimiter) or you may have forgotten to end your array environment with `\end{array}`

13.6 Can only be used in preamble.

Some commands, such as `\usepackage` may only appear in the preamble. Check to see where you have put it. For example, this error will be caused by doing:

```
\documentclass[a4paper]{article}

\begin{document}                    (incorrect)

\usepackage{graphicx}
```

instead of

```
\documentclass[a4paper]{article}

\usepackage{graphicx}               (correct)

\begin{document}
```

13.7 Command ... already defined

You have tried to define a command which already exists. Try giving it a different name. Remember never to redefine a command if you don't know what the command originally does.

Alternatively, you have tried to define an environment which already exists. Give the new environment a different name. Again, never redefine an environment where you don't know what the original environment does.

13.8 Display math should end with $$

You may have a dollar sign (`$`) in a `displaymath` or `equation` environment. Remember that `$` is short hand for `\begin{math}` or `\end{math}`, so you can't end one of the other environments with a `$`.

13.9 Environment ... undefined.

LaTeX doesn't recognise the environment you have specified.

- Check you have spelt the environment name correctly.

  You will get this error if you do, say,

  `\begin{docment}` (**incorrect**)

  instead of

  `\begin{document}` (**correct**)

- If it's your own environment, check you have defined the environment before using it.
- If the environment is defined in a package, check you have included the package using the `\usepackage` command.

13.10 Extra alignment tab has been changed to \cr

You have too many ampersands (`&`) in one row. The most probable cause is that you have forgotten the end of row command `\\` on the previous row. Remember also that if you have a `\multicolumn` command to span more than one column, you should have fewer `&`s in that row.

13.11 Extra \right

There are a number of possible causes. The most probable is that you have a `\right` that doesn't have a matching `\left`. (Remember left comes before right.) Another possible cause is that you have missed out `\end{array}`. (Remember that environments provide implicit grouping, and `\left` and its matching `\right` must appear within the same group level.)

13.12 File ended while scanning use of ...

The most usual cause of this error is a missing closing brace.

You will get this error if you do, say,

`\end{document` (**incorrect**)

instead of

`\end{document}` (**correct**)

13.13 File not found.

LaTeX can't find the file you have specified. You will be given the opportunity to type in the correct filename at the prompt. If you want to quit, simply type `X` followed by the return character ↵.

- Make sure that you have spelt the filename correctly.

  This error will be caused by, say,

  `\documentclass[a4paper]{artcle}` (**incorrect**)

instead of

`\documentclass[a4paper]{article}` (**incorrect**)

If this is the case, simply type in the correct name at the prompt (followed by the return character ↵)

- Make sure that the file is in the same directory as your document or in the LaTeX path. If the file is in another directory (not in the LaTeX path), you will need to specify the pathname, but remember that when using LaTeX under Windows, you need to use a forward slash (/) as the directory divider, as a backslash would be interpreted as a command. For example, if you have a file called `shapes.ps` in the subdirectory `pictures` then you would get a 'file not found' error message if you did

  `\includegraphics{shapes.ps}` (**incorrect**)

  instead of `\includegraphics{pictures/shapes.ps}` (**correct**)

- If the file is a package or class file, it's possible that you don't have that file installed on your computer. If this is the case you will need to download and install it as described in Section 6.2. Remember that you need to refresh the database after installing a new package or class file.

13.14 Illegal character in array arg

You have used a character in the argument of a `tabular` or `array` environment that is not allowed. The standard available characters are: `r` (right justified), `l` (left justified) and `c` (centred). This error will also occur if you have forgotten the argument to the `tabular` or `array` environment.

13.15 Illegal parameter number in definition

You have referred to a parameter (argument) number that is greater than the number of paramters you have specified. For example, suppose you defined the command to have only one parameter, then you can't use `#2` which refers to the second, non-existent, parameter. Remember that you need to specify how many parameters you want in the optional argument to `\newcommand`, otherwise it will be assumed that the command has no arguments.

13.16 Illegal unit of measure (pt inserted).

You have either not specified a unit when giving a length (even zero lengths must have a unit) or you have specified an invalid unit or you have misspelt the unit. Available units are listed in Table 12.1

13.17 Lonely \item

The command `\item` may only appear in one of the list making environments (such as `itemize`). Make sure you haven't forgotton your environment.

13.18 Misplaced alignment tab character &

You have used the special character `&` where you shouldn't have. Recall from Section 4.2 that if you want an & sign to appear you need to do `\&` not just `&`.

You would have got this error message if you had done, say,

`& our equipment` (**wrong**)

instead of

`\& our equipment` (**correct**)

13.19 Missing } inserted

You have missed a closing curly brace, or you may have missed out an argument.

Example: if the following line occurs in a tabular environment:

```
& \multicolumn{2}{c}\\
```

this will produce the error. (The third argument to `\multicolumn` has been omitted.)

13.20 Missing $ inserted

This message can be caused by a number of errors:

- You may have typed `$` instead of `\$` (you actually want a dollar symbol to appear). Recall from Section 4.2 that if you want a $ sign to appear you need to do `\$` not just `$`.

  You would have got this error message if you had done, say,

  `expenditure came to $2000.00` (**wrong**)

  instead of

  `expenditure came to \$2000.00` (**correct**)

- You might have missed the beginning of one of the mathematics environments (that is, you've used a command that must only appear in maths mode).

- You may have missed the end of a mathematics environment, or you may have a paragraph break within a `math`, `displaymath` or `equation` environment, which is not permitted. Make sure you don't have any blank lines within the environment. If you want a blank line in your code to make it easier to edit, try having a percent sign at the start of an empty line to ensure that the line is ignored by LaTeX. For example:

  ```
  \begin{equation}
  %
  E = mc^2
  %
  \end{equation}
  ```

13.21 Missing \begin{document}

You have put some text outside of the document environment. Check the following:

- You have remembered `\begin{document}`

  This error would be caused by, say,

  ```
  \documentclass[a4paper]{article}
  This is a simple document
  ```
  (**incorrect**)

  instead of

  ```
  \documentclass[a4paper]{article}
  \begin{document}
  This is a simple document
  ```
  (**correct**)

- You haven't placed any text before `\begin{document}`. For example:

  ```
  \documentclass[a4paper]{article}
  This is a simple document
  \begin{document}
  ```
  (**incorrect**)

  instead of

  ```
  \documentclass[a4paper]{article}
  \begin{document}
  This is a simple document
  ```
  (**correct**)

- You haven't missed out a backslash from either `\documentclass` or `\begin{document}`

  This error would be caused by, say,

  `documentclass[a4paper]{article}` (**incorrect**)

  instead of

  `\documentclass[a4paper]{article}` (**incorrect**)

13.22 Missing delimiter

You have forgotten to specify the type of delimiter you want (e.g. `( ) [ ] \{ \} | \| . `) (Remember to use a `.` if you want an invisible delimiter, and remember that if you want a curly brace, you must have a backslash followed by the curly brace.)

Example:

```
f(x) = \left{
\begin{array}{ll}
0 & x \leq 0\\
1 & x > 1
\end{array}
\right.
```
(**incorrect**)

instead of

```
f(x) = \left\{
\begin{array}{ll}
0 & x \leq 0\\
1 & x > 1
\end{array}
\right.
```

(**correct**)

13.23 Missing \endcsname inserted

This is a TEX error rather than a LATEX error which makes it harder to determine the cause, however it can be caused by placing a backslash in front of the name of an environment. (Remember that environment names do not contain a backslash.)

This error will be caused by, say,

`\begin{\sffamily}` (**incorrect**)

instead of

`\begin{sffamily}` (**correct**)

13.24 Missing \endgroup inserted

A number of things could have caused this. You may have missed out the end of an environment, or you may have an environment inside of another environment it's not allowed to be in. For example, this error can be caused by placing an `eqnarray` environment inside a `displaymath` environment, which is not allowed.

13.25 Missing number, treated as zero

LATEX is expecting a number. If your command takes more than one argument, check to make sure the arguments are in the correct order. For example, if you are using a `minipage` environment, you might have omitted the mandatory argument which specifies the width of the minipage, or you may have the optional arguments the wrong way round. The placement specifier should come first, followed by the height.

If you are using `\addtocounter` or `\setcounter` remember that the second argument must be a number, so if you want the value of a counter as the argument you must use `\value`. This error can be caused by, say,

`\setcounter{exercise}{chapter}` (**incorrect**)

instead of

`\setcounter{exercise}{\value{chapter}}` (**correct**)

13.26 Paragraph ended before \begin was complete

You've probably missed a closing brace at the end of the argument to `\begin`. This error will be caused by, say,

`\begin{document` (**incorrect**)

instead of

`\begin{document}` (**correct**)

13.27 Runaway argument

There are a number of possible causes of this error:

- Paragraph breaks are not permitted in many command arguments. You should use the corresponding environment if possible. For example, this error message will be generated by doing, say,

```
\textbf{This is a simple document.
Here is the first paragraph.

Here is the second paragraph.}
```
(**incorrect**)

  instead of

```
\begin{bfseries}
This is a simple document.
Here is the first paragraph.

Here is the second paragraph.
\end{bfseries}
```
(**correct**)

- The closing brace of a mandatory argument is missing: This error will be caused by, say,

  `\title{A Simple Document` (**incorrect**)

  instead of

  `\title{A Simple Document}` (**correct**)

- This error can also be caused by omitting the mandatory argument of an environment. For example:

```
\begin{thebibliography}
\bibitem{kopka95} A Guide to \LaTeX2e: document
```
(**incorrect**)

  instead of

```
\begin{thebibliography}{1}
\bibitem{kopka95} A Guide to \LaTeX2e: document
```
(**correct**)

13.28 Something's wrong–perhaps a missing \item.

You may have missed an `\item` command. The first object in a list environment must either be an `\item` command, or another list environment.

This error will be caused by, say,

```
\begin{itemize}
Animal
\item Vegetable
\item Mineral
\end{itemize}
```
(**incorrect**)

instead of

```
\begin{itemize}
\item Animal
\item Vegetable                                    (correct)
\item Mineral
\end{itemize}
```

This error can also be caused by a missing `\bibitem` in the bibliography. For example:

```
\begin{thebibliography}{1}
A Guide to \LaTeX2e: document                      (incorrect)
```

instead of

```
\begin{thebibliography}{1}
\bibitem{kopka95} A Guide to \LaTeX2e: document    (correct)
```

13.29 There's no line here to end.

You have placed a line breaking command (`\\`, `\newline` or `\linebreak`) where it doesn't make sense to have one.

13.30 Undefined control sequence

LaTeX doesn't understand the command you have used.

- Check to see if you have misspelt the command name (remember that all LaTeX command names are case-sensitive.)

  You will get this error if you do, say,

  `This is a simple \Latex\ document` (**incorrect**)

  instead of

  `This is a simple \LaTeX\ document` (**correct**)

- Check that you have remembered the space when typing `\ `. For example:

  `This is a simple \LaTeX\document` (**incorrect**)

  instead of

  `This is a simple \LaTeX\ document` (**correct**)

- If you are using a command that is defined in a package make sure you have included the package using `\usepackage`.

- Check that your command name hasn't run into the next piece of text. For example, you can do

  ```
  man{\oe}uvre
  ```

  or

  ```
  man\oe uvre
  ```

or

```
man\oe{}uvre
```

but not

```
man\oeuvre
```

- You have used a backslash instead of a forward slash as a directory divider. (Remember that when using LaTeX under Windows, you need to use a forward slash (/) as the directory divider, as a backslash would be interpreted as a command.)

  For example, suppose you have a file called `shapes.ps` in a subdirectory called `pictures`, then you would get an error if you did

  `\includegraphics{pictures\shapes.ps}` (**Incorrect**)

  instead of

  `\includegraphics{pictures/shapes.ps}` (**Correct**)

13.31 You can't use 'macro parameter character #' in horizontal mode

You have used the special character `#` where you shouldn't have. Recall from Section 4.2 that if you want a # sign to appear you need to do `\#` not just `#`.

This error message will be caused by doing, say,

`Item #1` (**Incorrect**)

instead of

`Item \#1` (**Correct**)

Bibliography

[1] "LaTeX : a document preparation system", Leslie Lamport, 2nd edition (updated for LaTeX2ε), Addison-Wesley (1994). (Cited on pages 2 and 62.)

[2] "A guide to LaTeX2ε: document preparation for beginners and advanced users", Helmut Kopka and Patrick W. Daly, Addison-Wesley (1995). (Cited on pages 2, 44, 55, 57, 62, 80, 87 and 102.)

[3] "The LaTeX companion", Michel Goossens, Frank Mittelbach and Alexander Samarin, Addison-Wesley (1994). (Cited on pages 2, 44, 55, 56, 57, 63, 80, 83, 87 and 102.)

[4] "The LaTeX graphics companion", Michel Goossens, Sebastian Rahtz and Frank Mittelbach, Addison-Wesley (1997). (Cited on pages 3, 63, 64 and 68.)

[5] "The LaTeX web companion", Michel Goossens and Sebastian Rahtz with Eitan Gurari, Ross Moore and Robert Sutor, Addison-Wesley (1999). (Cited on pages 3 and 11.)

[6] The TeX Archive. http://www.tex.ac.uk/ (Cited on pages 11, 19, 28, 68 and 131.)

I would strongly recommend that you have a look at the TeX Archive [6], particularly the Frequently Asked Questions and the On-Line Catalogue. It's also a good idea to look at the documentation that was installed with your TeX/LaTeX distribution, which will usually be located in the directory `c:\texmf\doc`, and the on-line help via the Start Menu:

Start → Programs → MiKTeX → Help

Index

Page numbers in italic indicate the principle definition.