An introduction to TEX and friends

Gavin Maltby

November 1992

Contents

List of Tables

Chapter 1

Getting acquainted with TeX

TeX is well known to be *the* typesetting package, and a vast cult of TeX lovers has evolved. But to the beginning TeX user, or to someone wondering if they should bother changing to TeX, it is often not clear what all the fuss is about. After all, are not both WordPerfect and Ventura Publisher capable of high quality output? Newcomers have often already seen what TeX is capable of (many books, journals, letters are now prepared with TeX) and so expect to find a tremendously powerful and friendly package. In fact they *do*, but that fact is well hidden in one's initial TeX experiences. In this chapter we describe a little of what makes TeX great, and why other packages cannot even begin to compete. Be warned that a little patience is required—TeX's virtues are rather subtle to begin with. But when the penny drops, you will wonder how you ever put up with anything different.

1.1 The spirit of TeX

In order to really appreciate TeX one needs to get a feel for what I call the "spirit" of TeX. When TeX appears to be making me work overtime to achieve something that I think ought to be perfectly straightforward, consultation with the TeX spirit shows me the error of my ways.

1.1.1 TEX is a typesetter, not a word-processor

TEX was designed with no limiting application in mind. It was intended to be able to prepare practically any document—from a single page all-text letter to a full blown book with huge numbers of formulae, tables, figures etc. The size and the complexity of a TEXable document is limited only by hardware considerations. Furthermore, TEX seeks to achieve all this whilst setting typesetting standards of the highest order for itself. The expertise of generations of professional printers has been captured in TEX, and it has been taught all the tricks of the trade.

Historically, printers prepared a document by placing metal characters in a large tray and arranging and binding them to form a page. This was very precisely done, but the ultimate precision was limited because of the mechanical nature of things and by time considerations. TEX prepares a page in an analogous manner (putting your characters and formulae into "boxes" which are then "glued" together to form the page), but has the advantage of enormous precision because placement calculations are performed by computer. Indeed, TEX's internal unit (the "scaled point") is about one-hundredth of the wavelength of natural light!

"But conventional word processors run on computers , too", you object. Yes, but their fundamental limitation is that they try to "keep up" with you and "typeset" your document as you type. This means that it can only make decisions at a local level (eg, it decides where to break a line just as you type the end of the line). TEX's secret is that it waits until you have typed the *whole* document before it typesets a single thing! This means that TEX can make decisions of a global nature in order to optimise the aesthetic appeal of your document. It has been taught what looks good and what looks bad (having been given a measure of the "badness" of various possibilities) and makes choices for your document that are designed to make it "minimally bad".

But TEX's virtues run much deeper than that, which is just as well because it is possible to get satisfactory, though imperfect, results from some word processors. One of TEX's strongest points is its ability to typeset complicated formulae with ease. Not only does TEX make hundreds of special symbols easily accessible, it will lay them out for you in your formulae. It has been taught all the spacing, size, font, ... conventions that printers have decided look best in typeset formulae. Although, of course, it doesn't understand any mathematics it knows the grammar of mathematics—it recognises binary relations, binary operators, unary operators, etc. and has been taught how these parts should be set. It is consequently rather difficult to get an equation to look bad in TEX.

Another advantage of compiling a document after it is typed is that cross-referencing can be done. You can label and refer back to chapters, sections, tables etc. by *name* rather than absolute number, and TEX will number and cross-reference these for you. Similarly, it will compile a table of contents, glossary, index and bibliography for you.

Essential to the spirit of TEX is that *it formats the document whilst you just take care of the content*, making for increased productivity. The cross-referencing just mentioned is just part of this. Many more labour-saving mechanisms are provided for through *style files*. These are generic descriptions of classes of documents, teaching TEX just how each class likes to be formatted. This is taught in terms of font preferences, default page sizes, placement of title, author, date, etc. For instance, a `paper` style file could teach TEX that when typesetting a theorem it should embolden the part that states the theorem number and typeset the text of the theorem statement in slanted Roman typeface (as in many journals). The typist simply provides and indication that a theorem is being stated, and then types the text of the theorem *without* bothering to choose any fonts or do any formatting—all that is done by the style file. Style files exist for all manner of document—letters, articles, papers, books, proceedings, review articles, and so on.

In addition to style files, there are *macro packages*. A *macro* is just a definition of a new TEX command in terms of existing ones. Don't think small when you think of macros! When typing a document that has a lot of repetition in it, say the same expression is used again and again in different different equations, you can define a macro in your document to abbreviate that expression. But macros can teach TEX how to typeset all sorts of complicated structures, not just parts of an equation. Many macro packages (files that are just collections of definitions) have been written to teach TEX all sorts of applications. There are specialist maths packages ($\mathcal{AMS}$-TEX, $\mathcal{AMS}$-LATEX), general purpose packages (LATEX), packages for setting tree diagrams, Feynmann diagrams, languages like Chinese, Arabic and Ancient Greek, orchestral scores, and many, many more. All these are freely available, a spin-off of the giant TEX cult.

Another facet of the design of TEX allows it to use practically *any* output device. In fact, TeX doesn't talk to any printers, screens, phototypesetters at all! Instead, when a document is compiled a *device independent* (`.dvi`) is produced—TeX does not compile with any particular output device in mind. Printer drivers are then invoked on this `.dvi` file and, in consultation with the font data for that printer, produce output suitable for the particular device. You can choose an HP Laserjet driver, or an Apple LaserWriter driver, or a dot matrix driver etc. All use the same `.dvi` file as input (and remember the material in there is set to enormous accuracy) and attempt to image that file on the particular device as faithfully as possible. If you are using a top of the line laser printer or phototypesetter, then TeX's massive internal precision will not be wasted. Alternatively, a dot matrix printer will give a coarse approximation of the ideal image that is suitable only for proof-reading. In addition to portability, these `.dvi` files help ensure that there are very few printing surprises when you move from one device to another: how many times has your favourite word-processor made you reformat a document when you wish to change printers?

There are many other motivations one could cite for the superiority of TeX. But it is time that we started to get our hands dirty. One last comment: TeX was not designed to supplant secretaries and professional printers—it was designed to aid them in their work and, in the words of the TeX designer Donald Knuth, allow them to "go forward and create masterpieces of the publishing art".It also allows those who generate the material to be typeset—mathematicians, physicists, computer scientists, etc—to prepare their own documents in a language that is intimately linked to the language we use for writing such material.

The novice reader will still have no idea of what a TeX source file looks like. Indeed, why do we keep referring to it as a *source file*? The fact of the matter is that TeX is essentially a *programming language*. Just as in any compiled language (e.g., Pascal, C) one prepares a source file and submits it to the compiler which attempts to produce an object file (`.dvi` file in the TeX case). To learn TeX is to learn the command syntax of the commands that can be used in the source file.

1.1.2 Typical TeX interfaces

TeX was designed to run on a multitude of computers. It is therefore the case that the documentation for TeX and its "friends" LaTeX, AMS-TeX, etc. is not computer specific. Only command syntax is described—i.e., the content of your source file—but few details of how to get from there to a printout are given. Those details are left to site-specific documents.

The average user loses little in using TeX on, say, a PC rather than on a bigger machine. Indeed, compilation times on the new PCs begin to rival those on a Sun Sparc Station 2 (no slouch). Running on top of DOS can cause memory problems when very large documents are being prepared. That aside, the quality of the document is not affected because of the careful design of TeX—whether you work on a a machine with massive floating point precision or a modest XT the `.dvi` files produced on compilation will be identical; and when those files are submitted to printer equivalent printer drivers (say for an HP LaserJet III attached to a Sun in one case and a PC in the other) the output will be identical because the font information they draw on is identical.

By the nature of TeX most time is spent editing the source document (before submitting it for compilation). No special interface is necessary here, you just use your favourite text editor (perhaps customising it to enhance TeXnical typing. Thus TeX user interfaces are usually small and simple, often even missing. One frequently uses TeX at command line level, just running the editor, compiler etc. as you need them. Sometimes a TeXshell program is present, which runs these for you when you choose various menu options.

Whatever the interface, there are just a few basic steps to preparing a document:

1. Choose a document style to base your document on (e.g., letter, article).

2. Glance through the material you have to type, and decide what definitions might be made to save you a lot of time. Also, decide on the overall structure of the prospective document (e.g., will the largest sectional unit be a chapter or a part?). If you are going to compose as you type, then pause a moment to think ahead and plan the structure of your document. The importance of this step cannot be overstressed, for it makes clear in *your* mind what you want from TeX.

3. Prepare your input file, specifying only the content and the logical structure (parts, sections, theorems,...) thereof and forgetting about formatting details.

4. Submit your input, or source, file to the TEX compiler for compilation of a `.dvi` file.

5. If the compiler finds anything in your source file strongly objectionable, say incorrect command syntax, then return to editing.

6. Run a *previewer* to preview your compiled document on the screen. Resolution is only limited by your screen, and can be very good indeed on some modern monitors.

7. Go back to editing your document until glaring errors have been taken care of.

8. Make a printout of your compiled document, and check for those errors that you failed to notice on the screen.

Performing these steps may be effected through typing at the system prompt (barebones technique) or through choosing menu options in a TEXshell program. The latter will probably provide some conveniences to make your life easier.

If you think this sounds like a lot of work, it is time that you consult with the TEX spirit! Sure your first couple of tries may be hesitant, but before long you'll find that you can take *less* time to prepare a document on TEX than on any other package.

Chapter 2

Getting started with LaTeX

2.1 Why start with LaTeX?

To answer this question we must say a little more about some of the macro packages we mentioned earlier.

The TeX typesetting system was designed by the eminent Stanford computer scientist Donald Knuth, on commission from the American Mathematical Society. It was designed with enormous care, to be ultimately powerful and maximally flexible. The enormous success of Knuth's design is apparent from the vast number of diverse applications TeX has found. In reading the following you must keep one thing clearly in mind: *there is only TeX language, and all the other packages whose names end in the suffix -TeX simply harness it's power via a whole lot of complicated macro definitions.*

TeX proper is a collection of around 300 so called *primitive* typesetting commands. These work at the very lowest level, affording enormous power. But to make this raw power manageable, some macros must be defined to tame raw TeX somewhat. The standard set of macros is called Plain TeX, and consists of about 600 macro definitions. It is clear that these definitions must be made in terms of TeX primitives, or in terms of previously made definitions. Plain TeX, however, is still no place for the timid. A strong working knowledge of TeX is still required to understand the ins and outs of Plain TeX.

In the few years after the initial TeX release (1982), the macro packages AMS-TeX and LaTeX were born. AMS-TeX was written by Michael Spivak, also on commission from the AMS. This package was designed to facilitate the preparation of the numerous books, journals, and review indices that fall under the auspices of the AMS and its affiliates. Married to the macro package was a style file—the AMS preprint style. This was distributed along with the macro package, so that authors submitting to journals could use it in the preparation of their articles. The given style was based on the style used by the *Journal of the American Mathematical Society*, i.e., it conformed to their page sizes and typographical conventions. This meant that people around the world produced papers that were all based on the same style. The clever part is this: when a source file is submitted to a journal *other* than the *Journal of the AMS*, the journal staff simply substitute their style file for the AMS preprint style and the paper will appear completely different *with no other changes to the source code*! To create their style file, a journal just needed to tweak the standard AMS prepint style: for instance, the original preprint style places author addresses at the very end of a paper; If a journal wishes this to appear on the first page then they just modify their in-house version of the style file, and the change will be effected without having to change the file submitted by the author.

LaTeX was written for more general usage. It lacks some of the mathematical finesse inherited by AMS-TeX from the vast experience of the AMS technical staff, but more than makes up for this in its ability to enhance the typesetting of letters, books, poetry, etc. LaTeX also scores high points for its enhanced command syntax.

With AMS-TeX and LaTeX being released at around the same time (1984–1985), there were born many AMS-TeX literate but LaTeX illiterate users, and conversely. LaTeX was easier to learn because of its more friendly syntax, and also provided powerful cross-referencing commands that AMS-TeX did not. So the AMS commissioned another project to furnish LaTeX users with the additional power of AMS-TeX while not compromising the LaTeX command syntax or cross-referencing commands. This resulted in the AMS-LaTeX macro package and associated style file for submission to journals.

That is why we will kick off our TeX careers with LaTeX! It is easier to learn and provides many conveniences, and the user who requires additional mathematical typesetting prowess can easily move on to AMS-LaTeX. Much of what we say will be true for TeX itself, but we shall regard LaTeX as the lowest common-denominator. By far the majority of LaTeX and AMS-LaTeX users will never have to learn "raw" TeX, for they will be shielded from direct exposure by the numerous powerful macro packages. In the rare case that something way out of the ordinary is required, the local TEX guru can be consulted.

2.2 LATEX formats, and we compose

The *free form nature* of the input file is essential to the spirit of TEX. As we type, we do not concern ourselves with linebreaks and pagebreaks so much as the content of what we are typing. In fact, we'll see that TEX will choose nice line breaks even for bizarre looking input. This is just part of the concept of only having to describe the *logical* structure of the document to LATEX, and not worry about nuisance-value formatting details. We inform LATEX of the logical structure of our document by telling it when to begin a new paragraph, subsection, section, chapter, theorem, definition, remark, poem, list etc. When typing a particular element of the logical structure, we need pay little attention to how we lay our source file out.

A consequence of this is that we have to go to a bit of effort to mess things up. Starting a new line, for instance, entails more than just pressing Return because LATEX will just regard the next word you type as exactly that—the next word in the paragraph. You have to specifically ask for a line to be terminated. Things like this may seem to be a bit of a nuisance, but it is a small price to pay for the automatic formatting that necessitated it. Further, such small inconveniences have been localised to rare events. I have, for instance, not once forced a new line up until this point in the present document.

2.3 Document styles

We have explained the concept of a document style during our discussion of the virtues of TEX and the discussion of AMS-TEX. It remains to name a few, and indicate where they would be used. One *always* has to choose a document style when preparing a document with LATEX.

The basic document styles in LATEX are `letter`, `article`, `report`, and `book`. Many more are available, but these few cover the majority of straightforward applications. This is because styles are not rigid—you can impose your own parameter choices if you want. So one chooses the style that most closely approximates the document you have in mind, and performs some minor tweaks here and there. The `article` style is used for documents that are to have the appearance of a journal or magazine article. The `report` style is usually used for larger documents than the `article` style. These styles really only differ in their choice of default page size, font, placement of title and author, sectional units, etc. and on how they format certain LaTeX constructs. You use the same LaTeX commands in each. Since the examples here will be small, we will choose to use the `article` document style.

There are a number of possible options with each document style. The syntax for choosing a document style follows. Don't worry if this leaves you with no idea of how to choose a document style, for we will soon be seeing some examples. Also, remember that an argument in square brackets is optional, and can omitted altogether (including the brackets).

`\documentstyle [`*options*`]{`*style*`}` where *style* is the main document style (eg `report`) and the optional argument *options* is a list of document style options chosen from the following list:

`11pt` chooses 11-point as the default font size for the document, instead of the default 10-point.

`12pt` chooses 12-point as the default font size.

`twoside` formats output as left and right pages, as in a book.

`twocolumn` produces two-column magazine like output.

`titlepage` applies to the `article` style only, causing the title and abstract to appear on a page each.

In fact there are many, many more document style options but we won't mention any more here.

2.4 Preparing a non-mathematical document

We assume that you have read the local guides to TeX at your site and have decided which system environment you want to work in. There you have been shown how to perform the steps required to produce a printed document from a LaTeX source file.

2.4.1 Sentences and paragraphs

Let's create our very first LaTeX document, which will consist of just a few paragraphs.

As mentioned above, paragraph input is free-form. You type the words and separate them by spaces so that LaTeX can distinguish between words. For these purposes, pressing Return is equivalent to inserting a space—it does not indicate the end of a line, but the end of a word. You tell LaTeX that a sentence has ended by typing a period followed by a space. LaTeX ignores extra spaces; typing three or three thousand will get you no more space between the words that these spaces separate than typing just one space. Finally, you tell LaTeX that a paragraph has ended by leaving one or more blank lines. In summary: LaTeX concerns itself only with the logical concepts end-of-word, end-of-sentence, and end-of-paragraph. Sounds complicated? An example should clear things up:

```
\documentstyle{article}
\begin{document}
Words within a sentence are ended by spaces.  One space
between words  is     equivalent      to any number. We are only
interested in separating      one    word      from        the
next, not in formatting      the space between them.
For these purposes, pressing Return
at the end of a line
and starting a new word on the next line
just serves to separate
words, not to cut a line short.
The end of a sentence is indicated by a period
followed by one or more spaces.

The end of a paragraph is indicated by leaving a blank line.
All this
means that we can type without too much regard for layout, and
the typesetter will sort things out for us.
\end{document}
```

produces the result

Words within a sentence are ended by spaces. One space between words is equivalent to any number. We are only interested in separating one word from the next, not in formatting the space between them. For these purposes, pressing Return at the end of a line and starting a new word on the next line just serves to separate words, not to cut a line short. The end of a sentence is indicated by a period followed by one or more spaces.

The end of a paragraph is indicated by leaving a blank line. All this means that we can type without too much regard for layout, and the typesetter will sort things out for us.

Perhaps you would like to try running LaTeX on the above input. Consult your local guide for details.

Note that we have learned more than just how LaTeX recognises words, sentences and paragraphs. We've also seen how to specify our choice of document style and how to tell LaTeX where our document begins and ends. Any material that is to be printed must lie somewhere between the declaration of `\begin{document}` and that of `\end{document}`. Definitions that are to apply to the entire document can be made before the declaration of the document beginning. The specification of document style must precede all other material.

In future examples we won't explicitly display the commands that select document style and delimit the start and end of the document. But if you wish to try any of the examples, don't forget to include those commands. The `article` document style will do for most of our examples. Of course, the preceding example looks not at all like an article because it is so short and because we specified no title or author information.

Most of what you need to know to type regular text is contained in the example above. When you consider that by far the majority of any document consists of straight text, it is obvious that LaTeX makes this fabulously straightforward. LaTeX will do all the routine work of formatting, and we simply get on with the business of composing.

LaTeX does more than simply choose pleasing line breaks and provide natural spacing when setting a paragraph. Remember we said that TeX has inherited the knowledge of generations of professional printers—well part of that knowledge includes being on the look-out for *ligatures*. These are combinations of letters within words which should be typeset as a single special symbol because they will "clash" with each if this is not done. Have a look at these words flight, flagstaff, chaff, fixation

and compare them with these

flight, flagstaff, chaff, fixation

See the difference? In the first set I let LaTeX run as it usually does. In the second I overruled it somewhat, and stopped it from creating ligatures. Notice how the 'fl', 'ff', and 'fi' combinations are different in the two sets—in the former they form a single symbol (a ligature) and in the latter they are comprised of two disjoint symbols. There are other combinations that yields ligatures, but we don't have to bother remembering any of them because LaTeX will take care of these, too.

Notice, too, that LaTeX has been taught how to hyphenate the majority of words. It will hyphenate a word if it feels that the overall quality of the paragraph will be improved. For long words it has been taught several potential hyphenation positions.

LaTeX also goes to a lot of trouble to try to choose pleasing page breaks. It avoids "widows", which are single lines of a paragraph occurring by themselves at either the bottom of a page (where it would have to be the first line of a paragraph) or at the top of a page (where it would have to be the last). It also "vertically justifies" your page so that all pages have exactly the same height, no matter what appears on them. As testimony to the success of the pagebreaking algorithm, I have (to this point) not once chosen a page break in this document.

2.4.2 Punctuation

Typists have a convention whereby a single space is left after a mid-sentence comma, and two spaces are left after a sentence-ending period. How do we enforce this if LaTeX treats a string of spaces just like a single one? The answer, unsurprisingly, is that we *don't*.

```
To have a comma followed by the appropriate space,  we simply
type a comma follows by at least one space. To end a sentence
we type a period with at least one following space.     No space will
be inserted if we type a comma or period followed straight away
by something other than a space, because there are times when
we won't require any space, i.e., we do what comes narurally.
```

will produce

> To have a comma followed by the appropriate space, we simply type a comma follows by at least one space. To end a sentence we type a period with at least one following space. No space will be inserted if we type a comma or period followed straight away by something other than a space, because there are times when we won't require any space, i.e., we do what comes naturally.

LaTeX will produce suitable space after commas, periods, semi-colons and colons, exclamation marks, question marks etc. if they are followed by a space. In stretching a line to justify to the right margin, it also knows that space after a punctuation character should be more "stretchable" than normal inter-word space and that space after a sentence-ending period should be stretched more than space after a mid-sentence comma. TeX knows the nature of punctuation if you stick to the simple rules outlined here. As we've already said, those rules tell LaTeX how to distinguish consecutive words, sentences, phrases, etc.

Actually, there is more to ending sentences than mentioned above. Since LaTeX cannot speak English, it works on the assumption that *a period followed by a space ends a sentence unless the period follows a capital letter.* This works most of the time, but can fail. To get a normal inter-word space after a period that doesn't end a sentence, follow the period by a *control space*—a `\␣` (a \ character followed by a space or return). Very rarely, you will have to force a sentence to end after a period that follows a capital letter (remember that LaTeX assumes this doesn't end a sentence). This is done by preceding the period with a `\@` command (you can guess from the odd syntax that this is rarely needed).

It's time we saw some examples of this. After all, this is our first experience of *control symbols* (don't worry, there are many more to come).

```
We must be careful not to confuse intra-sentence periods
with periods that end a sentence, i.e.\ we must remember
that our task is to describe the sentence structure.  Periods
that the typesetter requires a little help with typically result
from abbreviations, as in etc.\ and  others.  We have to work
somewhat harder to break a sentence after a capital letter,
but that shouldn't bother us to much if we keep up our intake
of vitamin E\@.  All this goes for other sentence-ending
punctuation characters, so I could have said vitamin E\@!
Fortunately, these are rare occurrences.
```

results in

> We must be careful not to confuse intra-sentence periods with periods that end a sentence, i.e. we must remember that our task is to describe the sentence structure. Periods that the typesetter requires a little help with typically result from abbreviations, as in etc. and others. We have to work somewhat harder to break a sentence after a capital letter, but that shouldn't bother us to much if we keep up our intake of vitamin E. All this goes for other sentence-ending punctuation characters, so I could have said vitamin E! Fortunately, these are rare occurrences.

Quotation marks is another area where LaTeX will do some work for us. Keyboards have the characters `` ` ``, `'`, and `"` but we want to to have access to each of ‘, ’, “, and ”. So we proceed like this:

```
`\LaTeX' is no conventional word-processor, and
to to get quotes, like ``this'', we type repeated
` and ' characters.  Note that modern
convention is that ``punctuation comes after
the closing quote character''.
```

which gives just what we want

> ‘LaTeX’ is no conventional word-processor, and to to get quotes, like “this”, we type repeated ` and ' characters. Note that modern convention is that “punctuation comes after the closing quote character”.

Very rarely, you have three quote characters together. Merely typing those three quote characters one-after-the-other is ambiguous—how should they be grouped? You tell LaTeX how you want them grouped by inserting a very small space called `\,`.

```
``\,`Green ham' or `Eggs?'\,'' is the question.
```

gives the desired result

> "'Green ham' or 'Eggs?'" is the question.

Since we have a typesetter at our disposal, we might as well use the correct dashes where we need them. There are three types of dash: the hyphen, the endash, and the emdash. A minus sign is not a dash.

Hyphens are typed as you'd hope, just by typing a `-` at the point in the word that you want a hyphen. Don't forget that LaTeX takes care of hyphenation that is required to produce pretty linebreaks. You only type a hyphen when you explicitly want one to appear, as in a combination like "inter-college".

An endash is the correct dash to use in indicating number ranges, as in "questions 1–3". To specify an endash you type two consecutive dashes on the keyboard, as in `1--3`.

An emdash is a punctuation dash, used at the end of a sentence—I tend to use them too much. To specify an emdash you type three consecutive dashes on the keyboard, as in "`...a sentence---I tend to...`".

```
Theorems 1--3 concern the semi-completeness
of our new construct---in the case that it
satisfies the first axiom.
```

yields

> Theorems 1–3 concern the semi-completeness of our new construct—in the case that it satisfies the first axiom.

2.4.3 Ties

When you always remember to use *ties*, you know that you are becoming TeXnically inclined. Ties are used to prevent LaTeX from breaking lines at certain places. LaTeX will always choose line breaks that result in the most aesthetically pleasing paragraph as judged by its stringent rules. But because LaTeX does not actually understand the material it is setting so beautifully, it can make some poor choices.

A *tie* is the character `~`. It behaves as a normal interword space in all respects *except* that the line-breaking algorithm will never break a line at that point. Thus

```
Dr. Seuss should be typed as Dr.~Seuss
```

for this makes sure that LaTeX will never leave the 'Dr' at the end of one line and put the 'Seuss' at the beginning of the next.

One should try to get in to the habit of typing ties first-time, not after waiting to see if LaTeX will make a poor choice This will allow you to make all sorts of changes to your text without ever having to go back and insert a tie at a point that has migrated to the end of a line from the middle of a line as a result of those changes. Remember, of course, that the line-breaks

Here are some more examples of places where you should remember to place ties.

```
Chapter~10      Donald~E. Knuth
Appendix~C      width~2
Figure~1        function~f
Theorem~2       1,~2, or~3
Lemmas 3 and~4  equals~5
```

2.4.4 Specially reserved symbols

In the sequel we will see that the the ten characters

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

are reserved for special use. Indeed, we have seen already that `\` and `~` are non-printing characters that perform special services (and we'll have a lot more to say about the use of `\`).

But what if we need one of these special symbols to appear in our document? The answer for seven of the symbols is to precede them by a `\` character, so forming another *control symbol* (remember that `\` followed by a space was also a control symbol).

```
It is not 100\% straightforward to typeset the
characters \$ \& \% \_ \{ \}, but given the
enormous convenience of the use they are normally
reserved for this is a small price to pay.
```

produces

> It is not 100% straightforward to typeset the characters $ & % _ { }, but given the enormous convenience of the use they are normally reserved for this is a small price to pay.

2.4.5 So what are control symbols and words?

In typing a document, we can think of ourselves as being in one of two distinct modes. We are either typing *literal text* (which will just be set into neat paragraphs for us) or we are typing text that will be *interpreted* by LaTeX as an instruction to insert a special symbol or to perform some action. Thus we are either typing material that will go straight into the document (with some beautification), or we are giving commands to LaTeX.

Some commands are implicit, in that we don't have to do anything much extra. For instance, we command LaTeX to end the present sentence by typing a period (that doesn't follow a capital letter). These are no so much commands as part of having to describe the logical structure of a document.

A *control word* is something of the form `\commandname`, where the command name is a word made up only of the letters a to z and A to Z. A *control symbol* consists of a \ followed by single symbol that is not a letter.

Here are some examples:

- we have met the control space symbol `\␣` before,
- the commands to set symbols like % and $ are control symbols
- `\@` was a control symbol that told told LaTeX that the very next period did really end the sentence,
- `\LaTeX` is a control word that tell LaTeX to insert its own name at the current point,
- `\clubsuit` instructs that a ♣ be inserted,
- `\pounds` inserts a £ symbol,
- `\S` inserts a ¶ symbol,
- `\em` makes the ensuing text *be emphasised*,

These examples show that control sequences can be used to access symbols not available from the keyboard, do some typesetting tricks like setting the word LaTeX the way it does, and change the appearance of whole chunks of text as with `\em`. We'll be meeting many more of these type of control sequences.

Another enormously powerful class of control sequences is those that accept *arguments*. They tell LaTeX to take the parts of text you supply and do something with them—like make a fraction by setting the first argument over the second and drawing a line of the appropriate length between them. These are part of what makes LaTeX so powerful, and here are some examples.

- `\chapter{The beginning}` causes LaTeX to start a new chapter with name "The Beginning", number it in sequence, typeset the chapter heading in a suitable font, and make an entry in the table of contents,
- `\overline{words}` causes $\overline{\text{words}}$ to be overlined,
- `\frac{a+b}{c+d}` sets the given two argument as a fraction, doing most of the dirty work for us: $\frac{a+b}{c+d}$,
- `\sqrt[5]{a+b}` typesets the fifth-root of $a+b$, like this: $\sqrt[5]{a+b}$. The `5` is in square brackets instead of braces because it is an optional argument and could be ommited all together (giving the default of square root),

Mandatory arguments are given enclosed by braces, and optional arguments enclosed by square brackets. Each command knows how many arguments to expect, so you don't have to provide any indication of that.

We have actually jumped the gun a little. The above examples include examples of *mathematical* typesetting, and we haven't yet seen how to tell LaTeX that it is typesetting maths as opposed to some other random string of symbols that it doesn't understand either. We'll come to mathematical typesetting in good time.

We need to dwell on a TeXnicality for a moment. How does LaTeX know where the name of a control sequence ends? Will it accept both `\pounds3` and `\pounds 3` in order to set £3, and will `\emWalrus` and `\em Walrus` both be acceptable in order to get *Walrus*? The answer is easy when you remember that a control word consists only of alphabetic characters, and a control symbol consists of exactly one nonalphabetic character.

So to determine which control sequence you typed, LaTeX does the following:

1. when a \ character is encountered, LaTeX knows that either a control symbol or a control word will follow.

2. If the \ is followed by a nonalphabetic character, then LaTeX knows that it is a control *symbol* that you have typed. It then recognises which one it was, typesets it, and goes on to read the character which follows the symbol you typed.

3. If the \ is followed by an alphabetic character, then LaTeX knows that it is a control word that you have typed. But it has to work out where the name of the control word ends and where the ensuing text takes over again. Since only alphabetic characters are allowed, LaTeX reads everything up to just before that first nonalphabetic character as the control sequence name. Since it is common to delimit the end of a control word by a space, LaTeX will *ignore* any space that follows a control word, since you want that space treated as end-of-control-word space rather than interword space.

This has one important consequence: The character in the input file immediately after a control symbol will be "seen" by LaTeX, but *any space following a control word will be discarded and never processed.* This does not affect one much if you adopt the convention of always typing a space after a control sequence name.

There is a rare circumstance where this necessitates a little extra work and thought, which we illustrate by example:

```
If we type a control word like \LaTeX in the running text
then we must be cautious, because the string of spaces that
come after it will be discarded by the \LaTeX\ system.
```

which produces the output

> If we type a control word like LaTeXin the running text then we must be cautious, because the string of spaces that come after it will be discarded by the LaTeX system.

2.4.6 Commands to change appearance

We've seen a little of how to access various symbols using control sequences and we mentioned the `\em` command to emphasise text, but we didn't see how to use them. We look here at commands that change the appearance of the text.

Each of the control words here is a directive rather than a control sequence that accepts an argument. This is because potential arguments consisting of text that wants to be emboldened or emphasised are very large, and it would be a nuisance to have to enclose such an argument in argument-enclosing braces.

To delimit the area of text over which one of these commands has effect (its *scope*) we make that text into what is called a *group*. Groups are used extensively in LaTeX to keep effects local to an area, rather than affecting the whole document. Apart from enhancing usability, this also in a sense protects distinct parts of a document from each other.

The LaTeX commands for changing type style are given in table 2.1, and those for changing type size are given in table 2.2. Commands for selecting fonts other than these are not discussed here.

`\rm`	Roman	`\it`	*italic*	`\sc`	CAPITALS
`\em`	*Emphasised*	`\sl`	*slanted*	`\tt`	`typewriter`
`\bf`	**boldface**	`\sf`	sans serif		

Table 2.1: Commands for selecting type styles

Each of the type style selection commands selects the specified style but does not change the size of font being used. The default type style is roman (you are reading a roman style font now). To change type size you issue one of the type size changing commands in table 2.2, which will select the indicated size in the currently active style. The release of LaTeX 3.0 (the present version is 2.09) will see the New Font Selection Scheme in place as a standard feature. This makes font matters much easier to deal with.

The point-size option referred to in table 2.2 is that specified in the `\documentstyle` command issued at the beginning of the input file. Through it you select that base (or default) font for your document to be 10, 11, or 12 point Roman. If no options are specified, the default is 10-point Roman. The table shows, for instance, that if I issue a `\large` in this document for

size	default (10pt)	11pt option	12pt option
`\tiny`	5pt	6pt	6pt
`\scriptsize`	7pt	8pt	8pt
`\footnotesize`	8pt	9pt	10pt
`\small`	9pt	10pt	11pt
`\normalsize`	10pt	11pt	12pt
`\large`	12pt	12pt	14pt
`\Large`	14pt	14pt	17pt
`\LARGE`	17pt	17pt	20pt
`\huge`	20pt	20pt	25pt
`\Huge`	25pt	25pt	25pt

Table 2.2: LaTeX size-changing commands.

which I chose the `12pt` document style option the result will be a 14-point Roman typeface.

We mentioned that to restrict the scope of a type-changing command we will set the text to be affected off in a group. Let's look at an example of this.

```
When we want to {\em emphasise\/} some text we
use the {\tt em} command, and use grouping to
restrict the scope.  We can change font {\large sizes}
in much the same way.  We can also obtain {\it italicised},
{\bf emboldened}, {\sc Capitals} and {\sf sans serif} styles.
```

When we want to *emphasise* some text we use the `em` command, and use grouping to restrict the scope. We can change font sizes in much the same way. We can also obtain *italicised*, **emboldened**, CAPITALS and sans serif styles.

Notice how clever grouping allows us to do all that without once having to use `\rm` or `\normalsize`.

One more thing slipped into that example—and italic correction `\/`. This is a very small amount of additional space that we asked to be inserted to allow for the change from sloping *emphasised* text to upright text, because the interword space has been made to look less substantial from the terminal sloping character. One has to keep an eye open for circumstances where this is necessary. See the effect of omitting an italic correction after the emphasised text earlier in this paragraph.

One might expect, by now, that LaTeX would insert an italic correction for us. But there are enough occasions when it is not wanted, and there is no good rule for LaTeX to use to decide just when to do it for us. So the italic correction is always left up to the typist.

2.4.7 Accents

LaTeX provides accents for just about all occasions. They are accessed through a variety of control symbols and single-letter control worlds which accept a single argument—the letter to be accented. These control sequences are detailed in table 2.3.

`\`{o}`	ò	(grave accent)
`\'{o}`	ó	(acute accent)
`\^{o}`	ô	(circumflex or "hat")
`\"{o}`	ö	(umlaut or dieresis)
`\~{o}`	õ	(tilde or "squiggle")
`\={o}`	ō	(macron or "bar")
`\.{o}`	ȯ	(dot accent)
`\u{o}`	ŏ	(breve accent)
`\v{o}`	ǒ	(háček or "check")
`\H{o}`	ő	(long Hungarian umlaut)
`\t{oo}`	o͡o	(tie-after accent)
`\c{o}`	o̧	(cedilla accent)
`\d{o}`	ọ	(dot-under accent)
`\b{o}`	o̱	(bar-under accent)

Table 2.3: Control sequences for accents

Thus we can produce ó by typing `\'{o}`, ǎ by typing `\v{a}`, and Pál Erdös by typing `P\'{a}l Erd\"{o}s`. Take special care when accenting an *i* or a *j*, for they should lose their dots when accented. Use the control words `\i` and `\j` to produce dotless versions of these letters. Thus the best way to type to type ĕxĭgent is `\u{e}x\u{\i}gent`.

2.4.8 Over-ruling some of TEX's choices

We've seen that ties can be used to stop linebreaks occurring between words. But how can we stop LATEX from hyphenating a particular word? More generally, how can we stop it from splitting any given group of characters across two lines. The answer is to make that group of characters appears as one solid *box*, through use of an `\mbox` command.

```
For instance, if we wanted to be sure that the word
{\em currentitem\/} is not split across lines
then we should type it as \mbox{\em currentitem}.
```

If for some reason we wish to break a line
in the middle of nowhere, preventing LATEX from justifying that line to the prevailing right margin, the we use the `\newline` command. One can also use the abbreviated form `\\`.

```
We start with a short line.\newline
And  now we continue with the normal
text, remembering that where we press
Return in the input file probably won't
correspond to a line break in the final
document.  More short lines\\
are easy, too.
```

will produce the line breaks we want

We start with a short line.
And now we continue with the normal text, remembering that where we press Return in the input file probably won't correspond to a line break in the final document. More short lines
are easy, too.

A warning is in order: `\newline` must only end part of a line that is "already set". It cannot be used to add additional space between paragraphs, nor to leave space for a picture you want to paste in. This is not to be awkward, but is just part of LATEX holding up its end of the deal by making you have to specially request additional vertical space. This prevents unwanted extra space from entering your document.

Later we shall see how to impose our own choice of page size, paragraph indentation, etc. For now we will continue to accept those declared for us in the document style.

2.4.9 Commenting your document

It is handy to be able to make comments to yourself in the source file for a document. Things like "I must rewrite this section" and "This is version 3 of the document" are likely. It would also be useful to be able to make the compiler ignore certain parts of the document at times. For this purpose we can use the `%` character, for all text on an input line that is after a `%` which is not part of an occurrence of the control symbol `\%` is discarded by the compiler. Here is an example:

```
There was a 100\% turnout today,
an all-time record.  %perhaps I should check this claim!
%Indeed, there are lots of unsubstantiated claims here!
This made for an extremely productive session.
```

will yields

> There was a 100% turnout today, an all-time record. This made for an extremely productive session.

2.4.10 Footnotes

Inserting footnotes is easy—LaTeX will position and number them for you. You just indicate *exactly* where the footnote marker should go, and provide the text of the footnote. The footnote text will be placed at the bottom of the present page in a somewhat smaller font, and be separated from the main text by a short horizontal rule[1] to conform with convention. The footnote in the last line was typed like this:

```
...rule\footnote{See for yourself! It's easy ...work.} to conform
```

No space was typed between the `rule` and the `\footnote`, because we want the footnote marker to appear right next to the last letter of the word.

Multiple footnotes[2] are obtained just by using the `\footnote` command again and again.

[1] See for yourself! It's easy when you don't have to do any work.
[2] Here is another footnote

2.4.11 Topmatter

We declare the title and author information using the `\title` and `\author` commands, each of which accept a single argument. Multiple authors are all listed in the argument of `\title`, separated by `\and`'s. The `\date` command can be used to date a document. After we have declared each of these, we issue a `\maketitle` command to have them typeset for us. In the `book` and `article` document styles this will result in a separate page; in the `article` style the "top matter" will be placed at the top of the first page. The style files determine the placement and the choice of font.

```
\title{A Thought for the Day}
\author{Fred Basset \and Horace Hosepipe}
\date{November 1992}
\maketitle
```

will produce something along the lines of

A Thought for the Day

Fred Basset   Horace Hosepipe

November 1992

This topmatter must appear *after* the `\begin{document}` and before any other printing material.

2.4.12 Sectioning commands

As part of our task of describing the logical structure of the document, we must indicate to LaTeX where to start sectional units. To do this we make use of the sectioning commands shown in table 2.4.

`\part`	`\subsection`	`\paragraph`
`\chapter`	`\subsubsection`	`\subparagraph`
`\section`		

Table 2.4: LaTeX sectioning commands

Each sectioning command accepts a single argument—the section heading that is to be used. LaTeX will provide the section numbering (and numbering of subsections within sections, etc.) so there is no need to include any number in the argument. LaTeX will also take care of whatever spacing is required to set the new logical unit off from the others, perhaps through a little extra space and using a larger font. It will also start a new page in the case that a new chapter is begun.

The `\part` command is used for major subdivisions of substantial documents. The `\paragraph` and `\subparagraph` commands are, unfortunately confusing. They are used to section off a modest number of paragraphs of text—they don't start new paragraphs (remember that that was done by leaving a blank line in our input file). The names were retained for historical reasons.

It is always a good idea to *plan* the overall sectional structure of a document in advance, or at least give it a little thought. Not that it would be difficult to change your mind later (you could use the global replace feature of an editor, for instance), but so that you have a good idea of the structure that you have to describe to LaTeX.

The sectioning command that began the present sectional unit of this document was

```
\subsection{Sectioning commands}
```

and that was all that was required to get the numbered section name and the table of contents entry.

There are occasions when you want a heading to have all the appearance of a particular sectioning command, but shouldn't be numbered as a section in its own right or produce a table of contents entry. This can be achieved through using the **-form* of the command, as in `\section*{...}`. We'll see that many LaTeX commands have such a *-form which modify their behaviour slightly.

Not only will LaTeX number your sectional units for you, it will compile a table of contents too. Just include the command `\tableofcontents` after the `\begin{document}` command and after the topmatter that should precede it.

2.4.13 LaTeX environments

Perhaps the most powerful and convenient concept in the LaTeX syntax is that of an *environment*. We will see most of the "heavy" typesetting problems we will encounter can be best tackled by one or other of the LaTeX environments.

Some environments are used to *display* a portion of text, i.e. to set it off from the surrounding text by indenting it. The `quote` and `verse` environments are examples of these. The `center` environment allows us to centre portions of text, while the `flushright` environment sets small portions of text flush against the right margin.

But the true power of LaTeX begins to show itself when we look at environments such as those that provide facilities for itemised or enumerated lists, complex tabular arrangements, and for taking care of figure and table positioning and captioning. What we learn here will also be applicable in typesetting some complicated mathematical arrangements in the next chapter.

All the environments are begun by a `\begin{`*name*`}` command and ended by an `\end{`*name*`}`, where *name* is the environment name. These commands also serve as begin-group and end-group[3] markers, so that all commands are local to the present environment—they cannot affect text outside the environment.

We can also have environment embedded within environment within environment and so on, limited only by memory available on the computer. We must, however, be careful to check that each of these *nested* environments is indeed contained within the one just outside of it.

2.4.14 `em` environment

We start with a very simple environment, one which provides an alternative to the `\em` command. Remember that `\em` does not accept an argument; it applies to everything within its scope as dictated by the group within which it is used. This can be tricky if we wish to emphasise a large amount of text, for we may forget a group-delimiting brace and so upset the entire grouping structure of our document. In cases where we fear this might happen, we can proceed as follows.

[3] See section 2.4.6

```
\begin{em}
We must always be careful to match our group-delimiting
braces correctly.  If the braces in a document are unevenly
matched then \LaTeX\ will become confused because we will
have, in effect, indicated different scopes than we
intended for commands.
\end{em}
```

which will give

> *We must always be careful to match our group-delimiting braces correctly. If the braces in a document are unevenly matched then* LaTeX *will become confused because we will have, in effect, indicated different scopes than we intended for commands.*

Although LaTeX doesn't care too much for how we format our source file, it is obviously a good idea to lay it out logically and readably anyway. This helps minimise errors as well as aids in finding them. For this reason I have adopted the convention of always placing the environment `\begin` and `\end` commands on lines by themselves.

2.4.15 `quote` and `quotation` environments

This environment can be used to display a part of a sentence or paragraph, or even several paragraphs, in such a manner that the material stands out from the rest of the text. This can be used to enhance readability, or to simply emphasise something. Its syntax is simple:

```
Horace smiled and retaliated:
\begin{quote}
\em You can mock the non-WYSIWYG nature of \TeX\
all you like.  You don't understand that that is
precisely what makes \TeX\ enormously more powerful
than that lame excuse for a typesetter you use.
And I'll bet that from start to finish of preparing
a document I'm quicker than you are, even if you
do type at twice the speed and have the so-called
advantage of WYSIWYG.  In your case, what you see
is {\em all\/} you get!
\end{quote}
and then continued with composing his masterpiece of the
typesetting art.
```

produces the following typeset material:

Horace smiled and retaliated:

> *You can mock the non-WYSIWYG nature of TEX all you like. You don't understand that that is precisely what makes TEX enormously more powerful than that lame excuse for a typesetter you use. And I'll bet that from start to finish of preparing a document I'm quicker than you are, even if you do type at twice the speed and have the so-called advantage of WYSIWYG. In your case, what you see is* all *you get!*

and then continued with composing his masterpiece of the typesetting art.

That is a much more readable manner of presenting Horace's piece of mind than embedding it within a regular paragraph. The `quote` environment was responsible for the margins being indented on both sides during the quote. This example has also been used to show how the commands that begin and end an environment restrict the scope of commands issued within that environment: The `\em` at the beginning of the quote did not affect the text following the quote. We have also learned here that if we use `\em` within already emphasised text, the result is roman type—and we don't require an italic correction here because the final letter of 'all' was not sloping to the right.

The `quotation` environment is used in just the same way as the `quote` environment above, but it is intended for setting long quotations of several paragraphs. It would be suitable for quoting a few paragraphs from the text of some speech, for instance. LaTeX treats the given text just like normal text that it has to set into paragraphs, except that it indents the margins a little.

2.4.16 `verse` environment

This is provided to facilitate the setting of poetry. When within the `verse` environment, we use `\newline` (or `\\`) to end a line; and what would normally signify a new paragraph serves to indicate the start of a new stanza. Let's have a shot at some cheap poetry.

```
\begin{verse}
Roses are red\\
Violets are blue\\
I think \TeX\ is great\\
And so will you!

Roses are still red\\
Violets are still blue\\
I'm schizophrenic\\
And so am I.
\end{verse}
```

will produce the following stunningly-creative "poem":

> Roses are red
> Violets are blue
> I think TeX is great
> And so will you!
>
> Roses are still red
> Violets are still blue
> I'm schizophrenic
> And so am I.

2.4.17 `center` environment

This environment allows the centring of consecutive lines of text, new lines being indicated by a `\\`. If you don't separate lines with the `\\` command then you'll get a centred paragraph the width of the page, which won't look any different to normal. If only one line is to be centred, then no **** is necessary.

```
The {\tt center} environment takes care of the vertical
spacing before and after it, so we don't need to leave any.
\begin{center}
If we leave no blank line after the\\
{\tt center} environment\\
then the ensuing text will not\\
be regarded as part of a new\\
paragraph, and so will not be indented.\\
\end{center}

In this case we left a blank line after the environment,
so the new text was regarded as starting a new paragraph.
```

gives the following text

The `center` environment takes care of the vertical spacing before and after it, so we don't need to leave any.

If we leave no blank line after the
`center` environment
then the ensuing text will not
be regarded as part of a new
paragraph, and so will not be indented.

In this case we left a blank line after the environment, so the new text was regarded as starting a new paragraph.

2.4.18 `flushright` and `flushleft` environments

The `flushright` environment causes each line to be set with its last character against the right margin, without trying to stretch the line to to current text width. The `flushleft` environment is similar.

```
We can stop \LaTeX\ from justifying each line to both the
left and the right margins.
\begin{flushright}
The {\tt flushright} environment is\\
used for text with an even right margin\\
and a ragged left margin.
\end{flushright}
\begin{flushleft}
and the {\tt flushleft} environment is\\
used for text with an even left margin\\
and a ragged right margin.
\end{flushleft}
```

gives the desired display

We can stop LaTeX from justifying each line to both the left and the right margins.

The <code>flushright</code> environment is
used for text with an even right margin
and a ragged left margin.

and the `flushleft` environment is
used for text with an even left margin
and a ragged right margin.

One must be wary not to lapse into "word-processing" mode when using these environments. Remember that pressing return at the end of a line in the input file does not serve to end the current line there, but just to indicate the end of another word. We have to use the `\\` command to end a line.

2.4.19 `verbatim` environment

We can simulate typed text using the `verbatim` environment. The `\tt` (typewriter text) type style can be used for simulating typed words, but runs into trouble if one of the characters in the simulated typed text is a specially reserved LaTeX character. For instance, `{\tt type \newline}` would not have the desired effect because LaTeX would interpret the `\newline` is an instruction to start a new line.

The `verbatim` environment allows the simulation of multiple typed lines.

Everything within the environment is typeset in typewriter font exactly as it appears in our source file—obeying spaces and line breaks as in the source file and not recognising the existence of any special symbols.

```
\begin{verbatim}
In the verbatim environment we can type anything
we like.
So we don't need to look out for uses of %, $, & etc,
nor will control sequences like \newline have any
effect.
\end{verbatim}
```

will produce the simulated input text

```
In the verbatim environment we can type anything
we like.
So we don't  need to look out for uses of %, $, & etc,
nor will control sequences like \newline have any
effect.
```

The only thing that cannot be typed in the `verbatim` environment is the sequence `\end{verbatim}`. You might notice that I still managed to simulate that control sequence above. One can always get what you want in TEX, perhaps with a little creativity.

If we want only to simulate a few typed words, such as when I say to use `\newline` to start a new line, then the `\verb` command is used. This command has a slightly odd syntax, pressed upon it by the use for which it was intended. It cannot accept an argument, because we may want to simulate typed text that is enclosed by `{braces}`. What one does is to choose any character that is *not* in the text to be simulated, and use a pair of these characters as "argument delimiters". I usually use the `@` or `"` charachters, as I rarely have any other uses for them. Thus

use `%` to obtain a % sign

is typed as

```
use \verb"\%" to obtain a \% sign
```

2.4.20 itemize, enumerate, description environments

LaTeX provides three predefined list-making environment, and a "primitive" list environment for designing new list environments of your own. We shall just describe the predefined ones here.

There is delightfully little to learn in order to be able to create lists. The only new command is `\item` which indicates the beginning of a new list item (and the end of the last one if this is not the first item). This command accepts an optional argument (which means you'd enclose it in square brackets) that can be used to provide an item label. If no optional argument is given, then LaTeX will provide the item label for you; in an `itemize` list it will bullet the items, in an `enumerate` list it will number the items, and in a list of `description`s the default is to have no label (which would look a bit odd, so you're expected to use the optional argument there).

Remember that `\item` is used to separate list items; it does not accept the list item as an argument.

```
\begin{itemize}
\item an item is begun with \verb@\item@
\item if we don't specify labels, then
      \LaTeX\ will bullet the items for us
\item I indent lines after the first in the
      input file, but that is just to keep things
      readable.  As always, \LaTeX\ ignores additional
      spaces.

\item a blank line between items is ignored, for
      \LaTeX\ is responsible for spacing items.
\item \LaTeX\ is in paragraph-setting mode when
      it reads the text of an item, and so will
      perform all the usual functions
\end{itemize}
```

produces the following itemised list:

- an item is begun with `\item`
- if we don't specify labels, then LaTeX will bullet the items for us
- I indent lines after the first in the input file, but that is just to keep things readable. As always, LaTeX ignores additional spaces.
- a blank line between items is ignored, for LaTeX is responsible for spacing items.
- LaTeX is in paragraph-setting mode when it reads the text of an item, and so will perform all the usual functions

Lists can also be embedded within one another, for they are just environments and we said that environments have this property. Remember that we must nest them in the correct order. We demonstrate with the following list, which also shows how to use the `enumerate` environment.

```
\noindent I still have to do the following things:
\begin{enumerate}
\item Sort out LAN accounts for people on the course
   \begin{itemize}
   \item Have new accounts created for those not already
         registered on the LAN
   \item Make sure all users have a personal directory
        on the data drive
   \item Give read and scan rights to users in the \TeX\
         directories
   \item Add users to the appropriate LAN print queues
   \end{itemize}
\item Have a \TeX\ batch file added to a directory that
      is on a public search path
\item Finish typing these course notes and proof-read them
\item Photocopy and bind the finished notes
\end{enumerate}
```

will give the following list

I still have to do the following things:

1. Sort out LAN accounts for people on the course
   - Have new accounts created for those not already registered on the LAN
   - Make sure all users have a personal directory on the data drive
   - Give read and scan rights to users in the TeX directories
   - Add users to the appropriate LAN print queues
2. Have a TeX batch file added to a directory that is on a public search path
3. Finish typing these course notes and proof-read them
4. Photocopy and bind the finished notes

See how I lay the source file out in a readable fashion. This is to assist myself, not LaTeX.

The `description` environment is, unsurprisingly, for making lists of descriptions.

```
\begin{description}
\item[\tt itemize] an environment for setting itemised lists.
\item[\tt enumerate] an environment for setting numbered lists.
\item[\tt description] an environment for listing descriptions.
\end{description}
```

will typeset the following descriptions:

`itemize` an environment for setting itemised lists.

`enumerate` an environment for setting numbered lists.

`description` an environment for listing descriptions.

Note that the scope of the `\tt` commands used in the item labels was restricted to the labels.

2.4.21 `tabbing` environment

This environment simulates tabbing on typewriters. There one chose the tab stops in advance (analysing the material to be typed for the longest item in each column) and typed entries consecutively, hitting the tab key to move to the next tab stop and return to move to the next line.

In the `tabbing` environment, we proceed similarly. We look for the worst-case line (that which will determine the desired tab stops) and use it to set the tabs by inserting `\=` control symbols at the points where we want tab stops. We then discard that line using `\kill`, since the worst-case line might not be the first line in the material we have to type. We then type each line, using `\>` to move to the next tab stop and `\\` to end a line.

```
\begin{tabbing}
Cheddar cheese \= Recommended \= \$2.00 \kill
Green Ham \> Recommended \> \$2.00
Eggs \> 1 a week \> \$1.50
Cheddar cheese \> Hmmm \> \$0.80
Yak cheese \> Avoid \> \$0.05
\end{tabbing}
```

gives the following uniformly-tabbed table

Green Ham	Recommended	\$2.00
Eggs	1 a week	\$1.50
Cheddar cheese	Hmmm	\$0.80
Yak cheese	Avoid	\$0.05

In the format line I chose the longest entry from each of the prospective columns. I lined some of the `\>` commands up in the source just to keep things readable.

Remember that excess spaces are ignored. LaTeX sets the `\kill`ed line normally and sees where the tab stops requested will be needed in the typeset text. Note also that commands given within the `tabbing` environment are local to *the current item.*

Actually, we use the above approach in the case that we require uniformly tabbed columns. The format line is not compulsory, and we can define tab stops dynamically. See if you can make sense of the following.

```
\begin{tabbing}
Entry in position 1,1 \= Entry 1,2 \= Entry 1,3\\
Entry in position 2,1 \> Entry 2,2 \> Entry 2,3\\
Entry 3,1 \= Entry 3,2 \> Entry 3,3\\
Entry 4,1 \> Entry 4,2 \> Entry 4,3
\end{tabbing}
```

which produces

Entry in position 1,1	Entry 1,2	Entry 1,3
Entry in position 2,1	Entry 2,2	Entry 2,3
Entry 3,1	Entry 3,2	Entry 3,3
Entry 4,1	Entry 4,2	Entry 4,3

The are additional commands that can be used within the tabbing environment to achieve special effects, but we won't be discussing them here.

2.4.22 `tabular` environment

The `tabular` environment is used to produce tables of items, particularly when the table is predominantly rectangular and when line drawing is required. LaTeX will make most decisions for us; for instance it will align everything for us without having to be told which are the longest entries in each column.

This environment is the first of many that use the TeX "tabbing character" `&`. This character is used to separate consecutive entries in a row of a table, array, etc. The end of a row is indicated in the usual manner, by using `\\`. In this way the individual cells of the table, or array, are clearly described to LaTeX, and it can analyse them to make typesetting decisions. Commands issued within a cell so defined are, again, local to that cell.

The `tabular` environment is also our first example of an *environment with arguments*. The arguments are given, in braces as usual, just after the closing brace after the environments name. In the case of `tabular` there is a single mandatory argument giving the justification of the entries in each column: `l` for left justified, `r` for right justified, and `c` for centred. There must be an entry for each column of the table, and there is no default. Let's start with a simple table.

```
\begin{tabular}{llrrl}
\bf Student name & \bf Number & \bf Test 1 & \bf Test 2 & \bf Comment\\
F. Basset    & 865432 & 78     & 85     & Pleasing\\
H. Hosepipe  & 829134 & 5      & 10     & Improving\\
I.N. Middle  & 853931 & 48     & 47     & Can make it
\end{tabular}
```

will produce the following no-frills table

Student name	**Number**	**Test 1**	**Test 2**	**Comment**
F. Basset	865432	78	85	Pleasing
H. Hosepipe	829134	5	10	Improving
I.N. Middle	853931	48	47	Can make it

Note that a `\\` was not necessary at the end of the last row. Also note that, once again, the alignment of the `&` characters was for human readability. It is conventional to set columns of numbers with right justification. The `\bf` directives apply only the entries in which they are given.

A `|` typed in the `tabular` environment's argument causes a vertical line to be drawn at the indicated position and extending for the height of the entire table. An `\hline` given in the environment draws a horizontal line extending the width of the table to be drawn at the vertical position at which the command is given. A `\cline`{*i*-*j*} draws a line spanning columns *i* to *j*, at the vertical position at which the command is given. A repeated line-drawing command causes a double line to be drawn. We illustrate line drawing in tables by putting some lines into our first table. We will type this example in a somewhat expanded form, trying to make it clear why the lines appear where they do.

```
\begin{tabular}{|l|l|r|r|l|}
\hline
\bf Student name & \bf Number & \bf Test 1 & \bf Test 2 & \bf Comment\\
\hline
F. Basset        & 865432     & 78         & 85         & Pleasing\\
\hline
H. Hosepipe      & 829134     & 5          & 10         & Improving\\
\hline
I.N. Middle      & 853931     & 48         & 47         & Can make it\\
\hline
\end{tabular}
```

which will give

Student name	Number	Test 1	Test 2	Comment
F. Basset	865432	78	85	Pleasing
H. Hosepipe	829134	5	10	Improving
I.N. Middle	853931	48	47	Can make it

That way of laying out the source file makes it clear where the lines will go. As we (by now) well know, the returns that we pressed after the `\\`s in typing this table might as well have been spaces as far as LaTeX is concerned. Thus it is common to have the `\hline` commands following the `\\`s on the input lines. We will do this in future examples.

The `\multicolumn` column can be used to overrule the overall format of the table for a few columns. The syntax of this command is

`\multicolumn` {*n*}{*pos*}{*item*}

where n is the number of columns of the original format that *item* is to span, and *pos* specifies the justification of the new argument.

```
\begin{tabular}{||l|c|c|c||} \hline
\multicolumn{4}{|c|}{\LaTeX\ size changing commands}\\ \hline
Style option         & 10pt (default) & \tt 11pt & \tt 12pt\\ \hline
\tt\bs footnotesize  & 8pt            & 9pt      & 10pt\\ \hline
\tt\bs small         & 9pt            & 10pt     & 11pt\\ \hline
\tt\bs large         & 12pt           & 12pt     & 14pt\\ \hline
\end{tabular}
```

produces the following table:

LaTeX size changing commands			
Style option	10pt (default)	`11pt`	`12pt`
`\footnotesize`	8pt	9pt	10pt
`\small`	9pt	10pt	11pt
`\large`	12pt	12pt	14pt

2.4.23 `figure` and `table` environments

Figures (diagrams, pictures, etc.) and tables (perhaps created with the `tabular` environment) cannot be split across pages. So LaTeX provides a mechanism for "floating" them to a nearby place where there is room for them. This may mean that your figure or table may appear a little later in the document than its declaration in the source file might suggest. You can suggest to LaTeX that it try to place the figure or table at the present position if there is room or, failing that, at the top or bottom of the present or following page. You can also ask for it to be presented by itself on a "page of floats".

You suggest these options to LaTeX through an optional argument to the environment. One lists a combination of the letters `h`, `t`, `b`, and `p` where

h means that the object should be placed *here* if there is room, so that things will appear in the same order as in the source file,

t means that the object can be placed at the *top* of the of a text page, but no earlier than the present page.

b means that the object can be placed at the *bottom* of a text page, but no earlier than the present page.

p means that the object should be set on a *page of floats* that consists only of tables and figures.

A combination of these indicates decreasing order of preference. The default is `tbp`. In this document I have tended to use `htbp`.

LaTeX will also number and caption a figure or table for you, and compile a list of tables and a list of figures. Just include `\listoffigures` and `\listoftables` next to your `\tableofcontents` command at the beginning of the document. To caption a table of figure, include `\caption{`*caption text*`}` just before the `\end{table}` or `\end{figure}` command. Here's a sample source file.

```
\begin{table}[htbp]
   \begin{tabular}{lrll}
   ...
   \end{tabular}
\caption{Mark analysis}
\end{table}
```

To leave space for a figure that will inserted by some other means at a later date, we can use the `\vspace` command:

```
\begin{figure}[htbp]
\vspace{9.5cm}
\caption{An artists impression}
\end{figure}
```

Including graphics files prepared with drawing packages is possible, but beyond the scope of this introduction.

2.4.24 The `letter` document style

All this and we still don't know how to prepare a simple letter! Actually, there is very little to it.

Figure 2.1 shows a sample letter. We declare our own address and signature before entering the `letter` environment because we can use repeated `letter` environments to prepare multiple letters from the same source file. The address of the intended recipient of a particular letter is given as an argument to the `letter` environment.

2.4.25 Common pitfalls; Error messages

By now it should be clear that we have to work quite accurately when preparing a document. Typing errors in the running text can be absorbed, but messing up a control sequence name will halt the compiler with an error message. Before we look at some common errors and some ways to avoid them, let's have a look at a sample error message.

You'll have noticed by now that when you run LaTeX on a source file, the transcript of the compiler session is written on a log file. When errors have accumulated to the point that LaTeX is hopelessly confused, it is time to debug your source file. The log file contains a reference to the line, or lines, of your source file that generated the error together with a description of the error.

TeX and LaTeX error messages appear frightening at first sight, to say the least. They are actually very informative, but they can take some getting used to. Mistyped control sequences cause little pain, but a missing `\end{`*environment*`}` can cause a good deal of confusion because it has the effect of making LaTeX try to set material into that environment that was never intended/designed to fit in such a place. Also, ommiting a mandatory argument can cause great confusion.

Suppose we type `\bold` instead of `\bf` in the following line:

```
this is going to be {\bold very} messy.
```

```
\documentstyle[12pt]{letter}
\begin{document}
\address{(Underneath) Lion Bridge\\
Midway down Commercial Road\\
Pietermaritzburg\\
3200}

\signature{F. Basset\\
Public nuisance}

\begin{Letter}{Director of Public Parks\\
Pietermaritzburg Municipality\\
Pietermaritzburg}

\opening{Kind Sir/Madam}

I wish to complain about the shocking practice of
fencing off the base of trees.  I notice with grave
concern that this has occurred in the park bordering
my stately residence.

This has already caused me great inconvenience
and public embarrassment, as you can imagine it would
for a hound of my social standing.  I demand that
you take these obscene obstructions away without
delay.

\closing{Yours anxiously}
\end{letter}
\end{document}
```

Figure 2.1: A sample letter

This produces the following error message:

```
! Undefined control sequence.
1.683 this is going to be {\bold
                                 very} messy.
?
```

That not so bad! The line beginning with `!` tells us that we have tried to used a control sequence that was not known to LaTeX; the `1.683` tells us that the error occurred on line 683 of the source file; and the error message is split over two lines with the break occurring at the point where LaTeX detected a problem.

But suppose we try the following

```
\begin{tabular}{llrrl}
Student name & Number & Test 1 & Test 2 & Comment\\
F. Basset    & 865432 & 78     & 85     & Pleasing\\
H. Hosepipe  & 829134 & 5      & 10     & Improving\\
I.N. Middle  & 853931 & 48     & 47     & Can make it

This shows that H.~Hosepipe's newfound concentration has...
```

i.e., we omit to provide the `\end{tabular}` that delimits the end of the environment. Not having been told that the environment is supposed to be concluded, LaTeX will try to set the text of the next paragraph as a table item—and will scream blue murder when it finds that it doesn't conform to the syntax demanded.

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

\@checkend ...urrenvir \else \@badend {#1}
                                          \fi
\enddocument ->\@checkend {document}
                                    \clearpage \begingroup \if@filesw...
\end #1->\csname end#1\endcsname
                                 \@checkend {#1}\expandafter \endgrou...
l.58 \end{document}

?
```

Now *that's* informative! Actually it *is* if we agree to ignore all but the the error indication line (the one beginning with the `!`) and the line telling us where LaTeX noticed that all was not well (the one beginning `l.58` in this case). The rest of the error message you can regard as being for your local TeX wizard to sort your problem out if you are unable to after consulting the manuals. Tough as it looks, we can decipher this message straight away: the error indicator line tell us that a `tabular` environment was ended incorrectly (in this case by an `\end{document}`).

TeX error messages aren't all that bad once you've made enough errors to get used to a few! Most can be avoided through *careful* preparation of the source file. Typing accurately and knowledge of the command syntax is a good start, but there are some other precautions that make good sense:

1. Even if LaTeX is happy with free-form input, try to lay your input file out as regularly and logically as possible. See our examples of environments for formats to adopt.

2. It is important that all group delimiters be properly matched, i.e., braces and `\begin{}`...`\end{}` must come in pairs. A good habit to fall in to is to always type such things in pairs and then move the cursor back between them and type the intervening material.

3. Don't forget command arguments when they are mandatory. Always ask yourself what a particular commands *needs from you* in order to make the decisions that are required of it.

4. Remember the 10 characters that are specially reserved for commenting, table item separation, etc.

5. When we look at mathematical typesetting in the next chapter, we will see that the same principles apply there.

6. Try to use a text editor that has a TEX mode, or at least one that will match brackets for you.

2.5 Summary

We have learned pretty much all we need to know in order to prepare non-mathematical documents. There has been quite a lot of material, all told, but we're fortunate that the average document requires only a fraction of what we've listed here. Furthermore, we'll find that what we've learned equips us with a good deal of the framework needed for mathematical typesetting.

The important thing to extract from this chapter is some feel for what I termed the "spirit of TEX" at the chapter beginning. I cannot emphasise enough the importance of getting your mind out of "word processing" mode and into "typesetting" mode. Always keep uppermost in your mind the task at hand: you are to describe the logical content of the document to LATEX, so furnishing it with enough information to perform all the formatting for you.

Many of the earlier sections of this chapter will become trivially easy to you after just a little experimentation with LATEX. The best way to learn the syntax of the more complicated environments is to use them—try typesetting the examples, for instance. It is important that you come to terms with the `tabular` environment, for its syntax is typical of many of the mathematical constructs that we will use.

If you have not already done so, then now is the time to try preparing some documents of your own. Try including all the material from this chapter, for that is the best way to remember it all. When the initial lack of familiarity wears off, you'll find that LATEX is really a whole lot friendlier and easier to use than you expected.

We must also recognise that there is a lot more to some of the command than detailed here. Some accept optional argumnets that were not mentioned, others have more options than we considered. And even once we have a full description of each command, there is still much to be learned for there is much that can be achieved through creative use of some of the environments.

Chapter 3

Mathematical typesetting with LaTeX

The last chapter taught us a good deal of what we need to know in order to prepare quite complicated non-mathematical documents. There are still a number of useful topics that we have not covered (such as cross-referencing), but we'll defer discussion of those until a later chapter. In the present chapter, we'll learn how LaTeX typesets mathematics. It should come as no surprise that LaTeX does most of the work for us.

3.1 Introduction

In text-only documents we saw that our task was to describe the logical components of each sentence, paragraph, section, table, etc. When we tell LaTeX to go into *mathematical mode*, we have to describe the logical parts of a formula, matrix, operator, special symbol, etc. TeX has been taught to recognize a binary operation, a binary relation, a variable, an operator that expects limits, and so on. We just need to supply the parts that make up each of these, and TeX will take care of the rest. It will leave appropriate space around operators, italicise variables, set an operator name in roman type, leave the correct space after colons, place sub- and superscripts in the correct positions (based on what it is you're working with), choose the correct typesizes, ... the list of things it has been taught is enormous. When you want to revert to setting normal text again, you tell LaTeX too leave maths mode and go back into the mode it was in (paragraphing mode).

LaTeX cannot be expected to perform these mode shifts itself, for it is not always clear just when it is mathematics that has been typed. For example, should an isolated letter `a` in the input file be regarded as a word (as in the definite article) or a mathematical variable (as in the variable a). There are no reliable rules for LaTeX to make such decisions by, so the begin-math and end-math mode switching is left entirely to you.

The symbol `$` is specially reserved[1] by LaTeX as the "math shift" symbol. When LaTeX starts setting a document it is in paragraphing mode, ready to set lines of the input file into paragraphs. It remains in this mode until it encounters a `$` symbol, which shifts LaTeX into mathematical mode. It now knows to be on the look-out for the components of a mathematical expression, rather than for words and paragraphs. It reads everything up to the next `$` sign in this mathemtical mode, and then shifts back to paragraphing mode (i.e. the mode it was in before we took it in to maths mode).

You must be careful to balance your begin-math and end-math symbols. It is often a good idea to type two `$` symbols and then move back between them and type the mathematical expression. If the math-shift symbols in a document are not matched, then LaTeX will become confused because it will be trying to set non-mathematical material as mathematics.

For those who find having the same symbol for both math-begin and math-end confusing or dangerous, there are two control symbols that perform the same operations: the control symbol `\(` is a begin-math instruction, and the control symbol `\)` is an end-math instruction. Since it is easy to "lose" a `$` sign when typing a long formula, a math environment is provided for such occasions: you can use `\begin{math}` and `\end{math}` as the math-shift instructions. Of course, you could just decide to use `$` and take your chances.

Let's have a look at some mathematics.

[1]See section 2.4.4

```
\LaTeX\ is normally in paragraphing mode, where
it expects to find the usual paragraph material.  Including
a mathematical expression, like $2x+3y - 4z= -1$, in the
paragraph text is easy.  \TeX\ has been taught to recognize
the basic elements of an expression, and typeset them appropriately,
choosing spacing, positioning, fonts, and so on.
Typing the above expression without entering maths
mode produces the incorrect result: 2x+3y - 4z= -1
```

will produce the following paragraph

> LaTeX is normally in paragraphing mode, where it expects to find the usual paragraph material. Including a mathematical expression, like $2x + 3y - 4z = -1$, in the paragraph text is easy. TeX has been taught to recognize the basic elements of an expression, and typeset them appropriately, choosing spacing, positioning, fonts, and so on. Typing the above expression without entering maths mode produces the incorrect result: 2x+3y - 4z= -1

Notice that LaTeX sets space around the binary relation $=$ and space around the binary operators $+$ and $-$ on the left hand side of the equation, ignoring the spacing we typed in the input. It was also able to recognize that the -1 on the right hand side of the equation was a unary minus—negating the 1 rather than being used to indicate subtraction—and so did not put space around it. It also italicised the variables x, y, and z. However, it did not italicise the number 1.

In typing a mathematical expression we must remember to keep the following in mind:

1. All letters that are not part of an argument to some control sequence will be italicised. Arguments to control sequences will be set according to the definition of the command used. So typing `$f(x)>0 for x > 1$` will produce

$$f(x) > 0 for x > 1$$

instead of the expression

$$f(x) > 0 \quad \mathrm{for} \quad x > 1$$

that we intended. Numerals and punctuation marks are set in normal roman type but LaTeX will take care of the spacing around punctuation symbols, as in

```
$f(x,y) \geq 0$
```

which produces

$$f(x,y) \geq 0 \quad .$$

2. Even a single letter can constitute a formula, as in "the constant a". To type this you enter `$a$` in your source file. If you do not go in to maths mode to type the symbol, you'll get things like "the constant a".

3. Some symbols have a different meaning when typed in maths mode. Not only do ordinary letters become variables, but symbols such as `-` and `+` are now interpreted as mathematical symbols. Thus in maths mode `-` is no longer considered a hyphen, but as a minus sign.

4. LaTeX ignores all spaces and carriage returns when in maths mode, without exception. So typing something like `the constant$ a$` will produce "the constant$a$". You should have typed `the constant $a$`. LaTeX is responsible for all spacing when in maths mode, and (as in paragraphing mode) you have to specially ask to have spacing changed. Even if LaTeX does ignore all spaces when in maths mode you should (as always in TeX) still employ spaces to keep your source file readable.

The above means that, at least for most material, a typist need not understand the mathematics in order to typeset it correctly. And even if one does understand the mathematics, LaTeX is there to make sure that you adhere to accepted typesetting conventions (whether you were aware of their existence or not).

So one could type either

```
$f(x, y)  =  2 (x+ y)y/(5xy - 3  )$
```

or

```
$f(x,y) = 2(x+y)y / (5xy-3)$
```

and you'd still get the correct result

$$f(x,y) = 2(x+y)y/(5xy-3) \quad .$$

There are some places where this can go wrong. For instance, if we wish to speak of the x-y plane then one has to know that it is an *endash* that is supposed to be placed between the x and the y, not a minus sign (as `$x-y$` would produce). But typing `$x--y$` will produce $x - -y$ since both dashes are interpreted as minus signs. To avoid speaking of the $x - y$ plane or the $x - -y$ plane, we should type it as `the $x$--$y$ plane`. We are fortunate that LaTeX can recognise and cope with by far the majority of our mathematical typesetting needs.

Another thing to look out for is the use of braces in an expression. Typing

```
${x : f(x)>0}$
```

will not produce any braces. This is because, as we well know, braces are reserved for delimiting groups in the input file. Looking back to section 2.4.4, we see how it should be done:

```
$ \{ x: f(x)>0 \} $
```

Math shift commands also behave as scope delimiters, so that commands issued in an expression cannot affect anything else in a document.

3.2 Displaying a formula

LaTeX considers an expression `$ ... $` to be word-like in the sense that it considers it to be eligible for splitting across lines of a paragraph (but without hyphenation, of course). LaTeX assigns quite a high penalty to doing this, thus trying to avoid it (remember that LaTeX tries to minimize the "badness" of a paragraph). When there is no other way, it will split the expression at a suitable place. But there are some expressions which are just too long to fit into the running text without looking awkward. These are best "displayed" on a line by themselves. Also, some expressions are sufficiently important that they should be made to stand out. These, too, should be displayed on a line of their own.

The mechanism for displaying an expression is the *display math* mode, which is begun by typing `$$` and ended by typing the same sequence (which again means that we'd better be sure to type them in pairs). Corresponding to the alternatives `\(` and `\)` that we had for the math shift character `$`, we may use `\[` and `\]` as the display-math shift sequences. One can also use the environment

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

which is equivalent to `$$ ... $$` and is suitable for use with long displayed expressions. If you wish LaTeX to number your equations for you you can use the environment

```
\begin{equation} ... \end{equation}
```

which is the same as the `displaymath` environment, except that an equation number will be generated.

It is poor style to have a displayed expression either begin a paragraph or be a paragraph by itself. This can be avoided if you agree to *never leave a blank line in your input file before a math display.*

We will see later how to typeset an expression that is to span multiple lines. For now, let's look at an example of displaying an expression:

```
For each $a$ for which the Lebesgue-set $L_a(f) \neq \emptyset$ we define
$$ % We could have used \begin{displaymath} here
{\cal B}_a(f) = \{ L_{a+r}(f) : r > 0  \},
$$ % and \end{displaymath} here
and these are easily seen to be completely regular.
```

which produces

For each a for which the Lebesgue-set $L_a(f) \neq \emptyset$ we define

$$\mathcal{B}_a(f) = \{L_{a+r}(f) : r > 0\},$$

and these are easily seen to be completely regular.

That illustrates how to display an expression, but also shows that we've got a lot more to learn about mathematical typesetting. Before we have a look at how to arrange symbols all over the show (e.g. the subscripting above) we must learn how to access the multitude of symbols that are used in mathematical texts.

3.3 Using mathematical symbols

LaTeX puts all the esoteric symbols of mathematics at our fingertips. They are all referenced by name, with the naming system being perfectly logical and systematic. None of the control words that access these symbols accepts an argument, but we'll soon see that some of them prepare LaTeX for something that might follow. For instance, when you ask for the symbol '$\sum$' LaTeX is warned that any sub- or superscripts that follow should be positioned appropriately as limits to a summation. In keeping with the TeX spirit, none of this requires any additional work on your part.

We'll also see that some of the symbols behave differently depending on where they are used. For instance, when I ask for $\sum_{i=1}^{n} a_i$ within the running text, the limits are places differently to when I ask for that expression to be displayed:

$$\sum_{i=1}^{n} a_i \quad .$$

Again, I typed nothing different here—just asked for display math mode.

It is important to note that *almost all of the special maths symbols are unavailable in ordinary paragraphing mode.* If you need to use one there, then use an in-line math expression `$...$`.

3.3.1 Symbols available from the keyboard

A small percentage of the available symbols can be obtained from just a single key press. They are $+ - = < > | / ( ) [ ]$ and $*$. Note that these must be typed *within maths mode* to be interpreted as math symbols.

Of course, all of a–z, A–Z, the numerals $0, 1, 2, \ldots, 9$ and the punctuation characters , ; and : are available directly from the keyboard. Alphabetic letters will be assumed to be variables that are to be italicised, unless told otherwise[2]. The numerals receive no special attention, appearing precisely as in normal paragraphing mode. The punctuation symbols are still set in standard roman type when read in maths mode, but a little space is left after them so that expressions like $\{x_i : i = 1, 2, \ldots, 10\}$ look like they should. Note that this means that normal sentence punctuation should not migrate into an expression.

[2]See section 3.4.6

3.3.2 Greek letters

Tables 3.1 and 3.2 show the control sequences that produce the letters of the Greek alphabet. We see that a lowercase Greek letter is simply is accessed by typing the control word of the same name as the symbol, using all lowercase letters. To obtain an uppercase Greek letter, simply capitalise the *first* letter of its name.

Just as `$mistake$` produces $mistake$ because the letters are interpreted as variables, so too will `$\tau \epsilon \chi$` produce the incorrectly spaced $\tau\epsilon\chi$ if you try to type greek words like this. TEX can be taught to set Greek, but this is not the way. τεχ, incidentally, is the Greek word for "art" and it is from the initials of the Greek letters constituting this word that the name TEX was derived. TEX is "the art of typesetting".

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

Table 3.1: Lowercase Greek letters

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

Table 3.2: Uppercase Greek letters

3.3.3 Calligraphic uppercase letters

The letters $\mathcal{A}, \ldots, \mathcal{Z}$ are available through use of the style changing command `\cal`. This command behaves like the other style changing commands `\em`, `\it`, etc. so its scope must be delimited as with them. Thus we can type

```
... for the filter $\cal F$ we have $\varphi({\cal F}) = \cal G$.
```

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

Table 3.3: Binary Operation Symbols

to obtain

for the filter $\mathcal{F}$ we have $\varphi(\mathcal{F}) = \mathcal{G}$.

There is no need to tabulate all the calligraphic letters, since they are all obtained by just a type style changing command. We will just list them so that we can see, for reference purposes, what they all look like. Here they are:

$$\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

3.3.4 Binary operators

LaTeX has been taught to recognise binary operators and set the appropriate space either side of one—i.e., it sets the first argument followed by a little space, then the operator followed by the same little space and finally the second argument. Table 3.3 shows the binary operators that are available via LaTeX control words (recall that the binary operators $+$, $-$, and $*$ can be typed from the keyboard). Here are some examples of their use:

Type	*To produce*
`$a+b$`	$a + b$
`$(a+b) \ otimes c$`	$(a + b) \otimes c$
`$(a \vee b) \wedge c$`	$(a \vee b) \wedge c$
`$X - (A \cap B) = (X-A) \cup (X-B)$`	$X - (A \cap B) = (X - A) \cup (X - B)$

3.3.5 Binary relations

LaTeX has been taught to recognize the use of binary relations, too. Table 3.4 shows those available via LaTeX control words. There are a few that you can obtain directly from the keyboard: $<$, $>$, $=$, and $|$.

To negate a symbol you can precede the control word that gives the symbol by a `\not`. Some symbols come with ready-made negations, which should be used above the `\not`'ing method because the slope of the negating line is just slightly changed to look more pleasing. Thus `\notin` should be used above `\not\in`, and `\neq` should be used above `\not =`.

If negating a symbol produces a slash whose horizontal positioning is not to your liking, then use the math spacing characters described in section 3.4.12 to adjust it.

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

Table 3.4: Binary relations

3.3.6 Miscellaneous symbols

Table 3.5 shows a number of general-purpose symbols. Remember that these are only available in maths mode. Note that `\imath` and `\jmath` should be used when you need to accent an i or a j in maths mode[3]—you cannot use `\i` or `\j` that were available in paragraphing mode. To get a prime symbol, you can use `\prime` or you can just type `'` when in maths mode, as in `$f''(x)=x$` which produces $f''(x) = x$.

[3] See section 3.3.10

$\aleph$	`\aleph`	$\prime$	`\prime`	$\forall$	`\forall`	∞	`\infty`
$\hbar$	`\hbar`	$\emptyset$	`\emptyset`	$\exists$	`\exists`	$\Box$	`\Box`
$\imath$	`\imath`	∇	`\nabla`	$\neg$	`\neg`	$\triangle$	`\triangle`
$\jmath$	`\jmath`	$\surd$	`\surd`	$\flat$	`\flat`	$\triangle$	`\triangle`
ℓ	`\ell`	$\top$	`\top`	$\natural$	`\natural`	$\clubsuit$	`\clubsuit`
$\wp$	`\wp`	$\bot$	`\bot`	$\sharp$	`\sharp`	$\diamondsuit$	`\diamondsuit`
$\Re$	`\Re`	$\Vert$	`|`	$\backslash$	`\backslash`	$\heartsuit$	`\heartsuit`
$\Im$	`\Im`	$\angle$	`\angle`	∂	`\partial`	$\spadesuit$	`\spadesuit`
$\mho$	`\mho`						

Table 3.5: Miscellaneous symbols

3.3.7 Arrow symbols

LaTeX has a multitude of arrow symbols, which it will set the correct space around. Note that vertical arrows can all be used as delimiters—see section 3.3.8. The available symbols are listed in table 3.6.

$\leftarrow$	`\leftarrow`	$\longleftarrow$	`\longleftarrow`	$\uparrow$	`\uparrow`
$\Leftarrow$	`\Leftarrow`	$\Longleftarrow$	`\Longleftarrow`	$\Uparrow$	`\Uparrow`
$\rightarrow$	`\rightarrow`	$\longrightarrow$	`\longrightarrow`	$\downarrow$	`\downarrow`
$\Rightarrow$	`\Rightarrow`	$\Longrightarrow$	`\Longrightarrow`	$\Downarrow$	`\Downarrow`
$\leftrightarrow$	`\leftrightarrow`	$\longleftrightarrow$	`\longleftrightarrow`	$\updownarrow$	`\updownarrow`
$\Leftrightarrow$	`\Leftrightarrow`	$\Longleftrightarrow$	`\Longleftrightarrow`	$\Updownarrow$	`\Updownarrow`
$\mapsto$	`\mapsto`	$\longmapsto$	`\longmapsto`	$\nearrow$	`\nearrow`
$\hookleftarrow$	`\hookleftarrow`	$\hookrightarrow$	`\hookrightarrow`	$\searrow$	`\searrow`
$\leftharpoonup$	`\leftharpoonup`	$\rightharpoonup$	`\rightharpoonup`	$\swarrow$	`\swarrow`
$\leftharpoondown$	`\leftharpoondown`	$\rightharpoondown$	`\rightharpoondown`	$\nwarrow$	`\nwarrow`
$\rightleftharpoons$	`\rightleftharpoons`	$\leadsto$	`\leadsto`		

Table 3.6: Arrow symbols

3.3.8 Expression delimiters

A pair of delimters often enclose an expression, as in

$$\left[\begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right] \quad \text{and} \quad f(x) = \left\{ \begin{array}{ll} x & \text{if } x < 1 \\ x^2 & \text{if } x \geq 1 \end{array} \right. .$$

LaTeX will scale delimiters to the correct size (determined by what they enclose) for you, if you ask it to. There are times when you don't want a delimiter to be scaled, so it is left up to you to ask for scaling.

To ask that a delimter be scaleable, you precede it by `\left` or `\right` according as it is the left or right member of the pair. Scaled delimiters must be balanced correctly. It sometimes occurs, as in the right-hand example above, that only one member of a delimiting pair is to be visible. For this purpose, use the commands `\left.` and `\right.` which will produce no visible delimiter but can be used to correctly balance the delimiters in an expression. For examples of the use of delimiters, see section 3.4.11 where we learn about arrays.

Table 3.7 shows the symbols that LaTeX will recognise as delimiters, i.e. symbols that may follow a `\left` or a `\right`. Note that you have to use `\left\{` and `\right\}` in order to get scaled braces.

<table>
<tr><td>$($</td><td>(</td><td>$)$</td><td>)</td><td>$\uparrow$</td><td><code>\uparrow</code></td></tr>
<tr><td>$[$</td><td>]</td><td>$]$</td><td>]</td><td>$\downarrow$</td><td><code>\downarrow</code></td></tr>
<tr><td>$\{$</td><td><code>\{</code></td><td>$\}$</td><td><code>\}</code></td><td>$\updownarrow$</td><td><code>\updownarrow</code></td></tr>
<tr><td>$\lfloor$</td><td><code>\lfloor</code></td><td>$\rfloor$</td><td><code>\rfloor</code></td><td>$\Uparrow$</td><td><code>\Uparrow</code></td></tr>
<tr><td>$\lceil$</td><td><code>\lceil</code></td><td>$\rceil$</td><td><code>\rceil</code></td><td>$\Downarrow$</td><td><code>\Downarrow</code></td></tr>
<tr><td>$\langle$</td><td><code>\langle</code></td><td>$\rangle$</td><td><code>\rangle</code></td><td>$\Updownarrow$</td><td><code>\Updownarrow</code></td></tr>
<tr><td>$/$</td><td>/</td><td>$\backslash$</td><td><code>\backslash</code></td><td></td><td></td></tr>
<tr><td>$\vert$</td><td>—</td><td>$\Vert$</td><td><code>\|</code></td><td></td><td></td></tr>
</table>

Table 3.7: Delimiters

3.3.9 Operators like $\int$ and $\sum$

These behave differently when used in display-math mode as compared with in-text math mode. When used in text, they will appear in their small form and any limits provided will be set so as to reduce the overall height of the operator, as in $\sum_{i=1}^{N} f_i$. When used in display-math mode, LaTeX will choose to use the larger form and will not try to reduce the height of the operator, as in

$$\sum_{i=1}^{N} f_i \quad .$$

Table 3.8 describes what variable-size symbols are available, showing both the small (in text) and the large (displayed) form of each. In section 3.4.1 we will learn how to place limits on these operators.

$\sum$ $\displaystyle\sum$	`\sum`	$\bigcap$ $\displaystyle\bigcap$	`\bigcap`	$\bigodot$ $\displaystyle\bigodot$	`\bigodot`
$\prod$ $\displaystyle\prod$	`\prod`	$\bigcup$ $\displaystyle\bigcup$	`\bigcup`	$\bigotimes$ $\displaystyle\bigotimes$	`\bigotimes`
$\coprod$ $\displaystyle\coprod$	`\coprod`	$\bigsqcup$ $\displaystyle\bigsqcup$	`\bigsqcup`	$\bigoplus$ $\displaystyle\bigoplus$	`\bigoplus`
$\int$ $\displaystyle\int$	`\int`	$\bigvee$ $\displaystyle\bigvee$	`\bigvee`	$\biguplus$ $\displaystyle\biguplus$	`\biguplus`
$\oint$ $\displaystyle\oint$	`\oint`	$\bigwedge$ $\displaystyle\bigwedge$	`\bigwedge`		

Table 3.8: Variable-sized symbols

3.3.10 Accents

The accenting commands that we learned for paragraphing mode do not apply in maths mode. Consult table 3.9 to see how to accent a symbol in maths mode (all the examples there accent the symbol u, but they work with any letter). Remember that i and j should lose their dots when accented, so `\imath` and `\jmath` should be used.

There also exist commands that give a "wide hat" or a "wide tilde" to their argument, `\widehat` and `\widetilde`.

$\hat{u}$	`\hat{u}`	$\acute{u}$	`\acute{u}`	$\bar{u}$	`\bar{u}`	$\dot{u}$	`\dot{u}`
$\check{u}$	`\check{u}`	$\grave{u}$	`\grave{u}`	$\vec{u}$	`\vec{u}`	$\ddot{u}$	`\ddot{u}`
$\breve{u}$	`\breve{u}`	$\tilde{u}$	`\tilde{u}`				

Table 3.9: Math accents

3.4 Some common mathematical structures

In this section we shall begin to learn how to manipulate all the symbols listed in section 3.3. Indeed, by the end of this section we'll be able to typeset some quite large expressions. In the section following this we will learn how use various alignment environments that allow us to prepare material like multi-line expressions and arrays.

3.4.1 Subscripts and superscripts

Specifying a sub- or superscript is as easy as you'd hope—you just give an indication that you want a sub- or superscript to the last expression and provide the material to be placed there, and LaTeX will position things correctly. So sub- and superscripting a single symbol, an operator, or a big array all involve the same input, and LaTeX places the material according to what the expression is that is being sub- or superscripted:

$$x^2 \quad , \quad \prod_{i=1}^{N} X_i \quad , \quad \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{array} \right]^2 \quad .$$

To tell LaTeX that you want a single character set as a superscript to the last expression, you just type a ^ before it. The "last expression" is the preceding group or, if there is no preceding group, the single character or symbol that the ^ follows:

Type	*To produce*
`$x^2$`	x^2
`$a^b$`	a^b
`$Y^X$`	Y^X
`$\gamma^2$`	γ^2
`$(A+B)^2$`	$(A+B)^2$
`$\left[ \frac{x^2+1}{x^2 + y^2} \right]^n$`	$\left[\frac{x^2+1}{x^2 + y^2} \right]^n$

Subscripts of a single character are equally easy—you just use the underscore character _ where you used ^ for superscripting:

Type	*To produce*
`$x_2$`	x_2
`$x_i$`	x_i
`$\Gamma_1(x)$`	$\Gamma_1(x)$

Now let's see how to set a sub- or superscript that consists of more than just one character. This is no more difficult than before if we remember the following rule: _ *and* ^ *set the group that follows them as a sub- and superscripts to the group that precedes the sub- and superscript symbols.* We see now now that our initial examples worked by considering a single character to be a group by itself. Here are some examples:

Type	*To produce*
`$a^2b^3$`	a^2b^3
`$2^{21}$`	2^{21}
`$2^21$`	2^21
`$a^{x+1}$`	a^{x+1}
`$a^{x^2+1}$`	a^{x^2+1}
`$(x+1)^3$`	$(x+1)^3$
`$\Gamma_{\alpha\beta\gamma}$`	$\Gamma_{\alpha\beta\gamma}$
`${}_1A_2$`	${}_1A_2$

In the very last example we see a case of setting a subscript to an empty group, which resulted in a kind of "pre-subscript". With some imagination this can be put to all sorts of uses.

In all of the above examples the sub- and superscripts were set to single-character groups. Nowhere did we group an expression before sub- or superscripting it. Even in setting the expression $(x+1)^3$, the superscript ^{3} was really only set to the character). If we had wanted to group the $(x+1)$ before setting the superscript, we would have typed `${(x+1)}^3$` which gives ${(x+1)}^3$, with the superscript slightly raised. One has to go to this trouble because, to most people, something like $(x^a)^b$ is just as acceptable and as readable as ${(x^a)}^b$. It also has the advantage of aligning the base lines in expressions such as

$$(ab)^{-2} = [(ab)^{-1}]^2 = [b^{-1}a^{-1}]^2 = b^{-1}a^{-1}b^{-1}a^{-1}$$

which looks more pleasing than if we use additional grouping to force

$${(ab)}^{-2} = {[{(ab)}^{-1}]}^2 = {[b^{-1}a^{-1}]}^2 = b^{-1}a^{-1}b^{-1}a^{-1} \quad ,$$

and the latter has rather more braces in it that require balancing.

Here are some more examples, showing how LaTeX will set things just as we want without any further work on our part:

Type	*To produce*
`$x^{y^z}$`	x^{y^z}
`$2^{(2^2)}$`	$2^{(2^2)}$
`$2^{2^{2^{\aleph_0}}}$`	$2^{2^{2^{\aleph_0}}}$
`$\Gamma^{z_c^d}$`	$\Gamma^{z_c^d}$

We can also make use of empty groups in order to stagger sub- and superscripts to an expression, as in

```
$\Gamma_{\alpha\beta}{}^{\gamma}{}_\delta$
```

which will yield

$\Gamma_{\alpha\beta}{}^{\gamma}{}_\delta$

One can specify the sub- and superscripts to a group in any order, but it is best to be consistent. The most natural order seems to be to have subscripts first, but you may think otherwise. It is also a good idea to always include your sub- and superscripts in braces (i.e. make them a group), whether they consist of just a single character or not. This enhances readability and also helps avoid the unfortunate case where you believe that a particular control word gives a single symbol yet it really is defined in terms of several.

3.4.2 Primes

LaTeX provides the control word `\prime` ($\prime$) for priming symbols. Note that it is not automatically at the superscript height, so that to get f' you would have to type

```
$f^\prime$ .
```

To make lighter work of this, LaTeX will interpret a right-quote character as a prime if used in maths mode. Thus we can type

```
$f'(g(x)) g'(x) h''(x)$
```

in order to get

$f'(g(x))g'(x)h''(x)$.

3.4.3 Fractions

LaTeX provides the `\frac` command that accepts two arguments: the numerator and the denominator (in that order). Before we look at examples of its use, let us just note that many simple in-text fractions are often better written in the form *num/den*, as with 3/8 which can be typed as `$3/8$`. This is also often the better form for a fraction that occurs *within* some expression.

Type	*To produce*
`$\frac{x+1}{x+2}$`	$\frac{x+1}{x+2}$
`$\frac{1}{x^2+1}$`	$\frac{1}{x^2+1}$
`$\frac{1+x^2}{x^2+y^2} + x^2 y$`	$\frac{1+x^2}{x^2+y^2} + x^2 y$
`$\frac{1}{1 + \frac{x}{2}}$`	$\frac{1}{1 + \frac{x}{2}}$
`$\frac{1}{1+x/2}$`	$\frac{1}{1+x/2}$

3.4.4 Roots

The `\sqrt` command accepts two arguments. The first, and optional, argument specifies what order of root you desire if it is anything other than the square root. The second, and mandatory, argument specifies the expression that the root sign should enclose:

Type	*To produce*
`$\sqrt{a+b}$`	$\sqrt{a+b}$
`$\sqrt[5]{a+b}$`	$\sqrt[5]{a+b}$
`$\sqrt[n]{\frac{1+x}{1+x^2}}$`	$\sqrt[n]{\frac{1+x}{1+x^2}}$
`$\frac{\sqrt{x+1}} {\sqrt[3]{x^3+1}}$`	$\frac{\sqrt{x+1}} {\sqrt[3]{x^3+1}}$

3.4.5 Ellipsis

Simply typing three periods in a row will not give the correct spacing of the periods if it is an ellipsis that is desired. So LaTeX provides the commands `\ldots` and `\cdots`. Centered ellipsis should be used between symbols like $+$, $-$, $*$, $\times$, and $=$. Here are some examples:

Type	*To produce*
`$a_1+ \cdots + a_n$`	$a_1+ \cdots + a_n$
`$x_1 \times x_2 \times \cdots \times x_n$`	$x_1 \times x_2 \times \cdots \times x_n$
`$v_1 = v_2 = \cdots = v_n = 0$`	$v_1 = v_2 = \cdots = v_n = 0$
`$f(x_1,\ldots,x_n) = 0$`	$f(x_1,\ldots,x_n) = 0$

3.4.6 Text within an expression

One can use the `\mbox` command to insert normal text into an expression. This command forces LaTeX temporarily out of maths mode, so that its argument will be treated as normal text. It's use is simple, but we must be wary on one count: remember that LaTeX ignores all space characters when in maths mode; so to surround words in an expression that were placed by an `\mbox` command by space you must include the space in the `\mbox` argument.

Type	*To produce*
`$f_i(x) \leq 0 \mbox{ for } x \in I$`	$f_i(x) \leq 0 \mbox{ for } x \in I$
`$\Gamma(n)=(n-1)! \mbox{ when $n$ is an integer}$`	$\Gamma(n)=(n-1)! \mbox{ when n is an integer}$

In section 3.4.12 we'll learn of some special spacing commands that can be used in math mode. These are often very useful in positioning text within an expression, enhancing readability and logical layout.

3.4.7 Log-like functions

There are a number of function names and operation symbols that should be set in normal (roman) type in an expression, such as in

$$f(\theta) = \sin\theta + \log(\theta+1) - \sinh(\theta^2+1)$$

and

$$\lim_{h\to 0}\frac{\sin h}{h} = 1 \quad .$$

We know that simply typing `$log\theta$` would produce the incorrect result

$$log\theta$$

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

Table 3.10: Log-like functions

and that using `$\mbox{log}\theta$` would leave us having to insert a little extra space between the log and the θ

$$\mbox{log}\theta \quad .$$

So LaTeX provides a collection of "log-like functions" defined as control sequences. Table 3.10 shows those that are available. Here are some examples of their use:

Type	*To produce*
`$f(x)=\sin x + \log(x^2)$`	$f(x)=\sin x + \log(x^2)$
`$\delta = \min \{ \delta_1, \delta_2 \}$`	$\delta = \min \{ \delta_1, \delta_2 \}$
`$\chi(X) = \sup_{x\in X} \chi(x)$`	$\chi(X) = \sup_{x\in X} \chi(x)$
`$\lim_{n \rightarrow \infty} S_n = \gamma$`	$\lim_{n \rightarrow \infty} S_n = \gamma$

Notice how LaTeX does more than just set an operation like sup in roman type. It also knew where a subscript to that operator should go.

3.4.8 Over- and Underlining and bracing

The `\underline` command will place an unbroken line under its argument, and the `\overline` command will place an unbroken line over its argument. These two commands can also be used in normal paragraphing mode (but be careful: LaTeX will not break the line within an under- or overlined phrase, so don't go operating on large phrases).

You can place horizontal braces above or below an expression by making that expression the argument of `\overbrace` or `\underbrace`. You can place a label on an overbrace (resp. underbrace) by superscripting (resp. subscripting the group defined by the bracing command.

Type	*To produce*
`$\overline{a+bi} = a- bi$`	$\overline{a+bi} = a - bi$
`$\overline{\overline{a+bi}} = a+bi$`	$\overline{\overline{a+bi}} = a + bi$

And some examples of horizontal bracing:

```
$A^n=\overbrace{A \times A \times \ldots \times A}^{\mbox{$n$ terms}}$

$\forall x \underbrace{\exists y (y \succ x)}_{\mbox{scope of $\forall$}}$
```

will produce

$$A^n = \overbrace{A \times A \times \ldots \times A}^{\mbox{n terms}}$$

and

$$\forall x \underbrace{\exists y (y \succ x)}_{\mbox{scope of $\forall$}}$$

3.4.9 Stacking symbols

LaTeX allows you to set one symbol above another through the `\stackrel` command. This command accepts two arguments, and sets the first centrally above the second.

Type	*To produce*
`$X \stackrel{f^*}{\rightarrow}Y$`	$X \stackrel{f^*}{\rightarrow} Y$
`$f(x) \stackrel{\triangle}{=} x^2 + 1$`	$f(x) \stackrel{\triangle}{=} x^2 + 1$

3.4.10 Operators; Sums, Integrals, etc.

Each of the operation symbols in table 3.8 can occur with limits. They are specified as sub- and superscripts to the operator, and LaTeX will position them appropriately. In an in-text formula they will appear in more-or-less the usual scripting positions; but in a displayed formula they will be set below and above the symbol (which will also be a little larger). The following should give you an idea of how to use them:

Type	*To produce*
`$\sum_{i=1}^{N} a_i$`	$\sum_{i=1}^{N} a_i$
`$\int_a^b f$`	$\int_a^b f$
`$\oint_{\cal C}f(x)\,dx$`	$\oint_{\cal C}f(x)\,dx$
`$\prod_{\alpha \in A} X_\alpha$`	$\prod_{\alpha \in A} X_\alpha$
`$\lim_{N\rightarrow\infty}\sum_{i=1}^{N}f(x_i)\Delta x_i$`	$\lim_{N\rightarrow\infty}\sum_{i=1}^{N}f(x_i)\Delta x_i$

We'll have more to say about the use of `\,` in section 3.4.12. Let's have a look at each of those expressions when displayed:

$$\sum_{i=1}^{N} a_i \quad , \quad \int_a^b f \quad , \quad \oint_{\cal C}f(x)\,dx \quad , \quad \prod_{\alpha \in A} X_\alpha \quad , \quad \lim_{N\rightarrow\infty}\sum_{i=1}^{N}f(x_i)\Delta x_i$$

3.4.11 Arrays

The `array` environment is provided for typesetting arrays and array-like material. It accepts two arguments, one optional and one mandatory. The optional argument specifies the vertical alignment of the array—use `t`, `b`, or `c` to align the top, bottom, or centre of the array with the centreline of the line it occurs on (the default being `c`). The second argument is as for the `tabular` environment: a series of `l`, `r`, and `c`'s that specify the number of columns and the justification of these columns. The body of the `array` environment uses the same syntax as the `tabular` environment to specify the individual entries of the array.

For instance the input

```
... let $A = \begin{array}{rrr}
12 & 3 & 4\\
-2 & 1 & 0\\
3 & 7 & 9
\end{array}$ ...
```

will produce the output

$$\text{let } A = \left[\begin{array}{rrr}
12 & 3 & 4\\
-2 & 1 & 0\\
3 & 7 & 9
\end{array}\right]$$

Note that we had to choose and supply the enclosing brackets ourselves (they are not placed for us so that we can use the `array` environment for array-like material; also, we get to choose what type of brackets we want this way). As in the `tabular` environment, the scope of a command given inside a matrix entry is restricted to that entry.

We can use ellipsis within arrays as in the following example:

```
\det A = \left| \begin{array}{cccc}
a_{11} & a_{12} & \cdots & a_{1n}\\
a_{21} & a_{22} & \cdots & a_{2n}\\
\vdots & \vdots & \ddots & \vdots\\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{array} \right|
```

which produces

$$\det A = \left| \begin{array}{cccc}
a_{11} & a_{12} & \cdots & a_{1n}\\
a_{21} & a_{22} & \cdots & a_{2n}\\
\vdots & \vdots & \ddots & \vdots\\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{array} \right|$$

The `array` environment is often used to typeset material that is not, strictly speaking, an array:

```
f(x) = \left\{ \begin{array}{ll}
   x & \mbox{for $x<1$}\\
 x^2 & \mbox{for $x \geq 1$}
\end{array} \right.
```

which will yield

$$f(x) = \left\{ \begin{array}{ll}
x & \mbox{for $x<1$}\\
x^2 & \mbox{for $x \geq 1$}
\end{array} \right.$$

3.4.12 Changes to spacing

Sometimes LaTeX needs a little help in spacing an expression, or perhaps you think that the default spacing needs adjusting. For these purposes we have the following spacing commands:

`\,`	thin space	`\:`	medium space
`\!`	negative thin space	`\;`	thick space
`\quad`	a quad of space	`\qquad`	two quads of space

The spacing commands `\,`, `\quad`, and `\qquad` can be used in paragraphing mode, too. Here are some examples of these spacing commands used to make subtle modifications to some expressions.

Type	*To produce*
`$\sqrt{2} \, x$`	$\sqrt{2} \, x$
`$\int_a^b f(x)\,dx$`	$\int_a^b f(x)\,dx$
`$\Gamma_{\!2}$`	$\Gamma_{\!2}$
`$\int_a^b \! \int_c^d f(x,y)\,dx\,dy$`	$\int_a^b \! \int_c^d f(x,y)\,dx\,dy$
`$x / \! \sin x$`	$x / \! \sin x$
`$\sqrt{\,\sin x}$`	$\sqrt{\,\sin x}$

3.5 Alignment

Recall that the `$equation$` environment can be used to display and automatically number a single- line equation[4]. The `eqnarray` environment is used for displaying and automatically numbering either a single expression that spreads over several lines or multiple expressions, while taking care of alignment for us. The syntax is similar to that of the `tabular` and `array` environments, except that no argument is necessary to declare the number and justification of columns. The `eqnarray*` environment does this without numbering any equations.

```
\begin{eqnarray}
(a+b)(a+b) & = & a^2 + 2ab + b^2\\
(a+b)(a-b) & = & a^2 - b^2
\end{eqnarray}
```

will give

$$(a+b)(a+b) \quad = \quad a^2 + 2ab + b^2 \tag{3.1}$$
$$(a+b)(a-b) \quad = \quad a^2 - b^2 \tag{3.2}$$

[4] See section 3.2

See how we identify the columns so as to line the = signs up. We can also leave entries empty, to obtain effect like the following:

```
\begin{eqnarray*}
\frac{d}{dx} \sin x & = & \lim_{h\rightarrow0}\frac{\sin(x+h)-\sin x}{h}\\
 & = & \lim_{h\rightarrow0}\frac{\sin x\cos h + \cos x\sin h - \sin x}{h}\\
 & = & \lim_{h\rightarrow0}\frac{\sin x(\cos h-1)}{h} + \cos x\frac{\sin h}{h}\\
 & = & \cos x
\end{eqnarray*}
```

which produces

$$\begin{aligned}\frac{d}{dx}\sin x &= \lim_{h\rightarrow 0}\frac{\sin(x+h)-\sin x}{h}\\ &= \lim_{h\rightarrow 0}\frac{\sin x\cos h+\cos x\sin h-\sin x}{h}\\ &= \lim_{h\rightarrow 0}\left\{\frac{\sin x(\cos h-1)}{h}+\cos x\frac{\sin h}{h}\right\}\\ &= \cos x\end{aligned}$$

3.6 Theorems, Propositions, Lemmas, ...

Suppose you document contains four kinds of theorem-like structures: "theorems", "propositions", "conjectures", and "wild guesses". Then near the beginning of the document you should have something like the following:

```
\newtheorem{thm}{Theorem}
\newtheorem{prop}{Proposition}
\newtheorem{conjec}{Conjecture}
\newtheorem{wildshot}{Hypothesis} % make it sound good!
```

The first argument to `\newtheorem` defines a new theorem-like environment name of your own choosing. The second argument contains the text that you want inserted when your theorem is proclaimed:

```
\begin{thm} $X$ is normal if, and only if, each pair of disjoint
closed sets in $X$ is completely separated.
\end{thm}

\begin{wildshot} % remember, we chose the name 'wildshot'
The property of Moore extends to all objects of the class $\Sigma$.
\end{wildshot}
```

which will produce the following:

Theorem 1 *X is normal if, and only if, each pair of disjoint closed sets in X is completely separated.*

Hypothesis 1 *The property of Moore extends to all objects of the class Σ.*

Notice that LaTeX italicises the theorem statement, and that you still have to shift in to maths mode when you want to set symbols and expression. Typically, it is the style file that determines what a theorem will appear like—so don't go changing this if you are preparing for submission for publication (because the journal staff want to substitute their production style for your document style choice, and not be over-ridden by other commands).

3.7 Where to from here?

We have covered a good deal of LaTeX's mathematical abilities, albeit rather superficial coverage here and there. There is much that has been impressive, but there is clearly a lot more to TeXnical typesetting than we have covered here—it is not difficult to think of an expression that we don't yet know how to typeset. Also, there are places where LaTeX is a little weak and it leaves us to do somewhat more work than the spirit of TeX would suggest.

Of course, we cannot criticise LaTeX until we know its full capability. So the first place to go from here is the LaTeX *User's Guide & Reference Manual*. Particularly, the command reference guide in Appendix C of that book is an invaluable source of LaTeX information that few can afford to do without. With good knowledge of the LaTeX environments and their options (and we've left out many here) one can accomplish a good deal of most typesetting problems. A little imagination (say putting an environment to a slightly non-standard use) can often solve more difficult problems. Lastly, of course, much of raw TEX still sits underneath LATEX and so it is true to say that you can do *anything* with LATEX—but you may need some divine inspiration from time to time (ask your local TEX guru).

In the next chapter we will look, very briefly, at a number of LATEX commands that we have not yet considered. Nothing exciting on the mathematical front, but there is some other important material (e.g. cross-referencing and page-sizing). For now, let's look at the "way forward" with respect to mathematical typesetting.

3.8 AMS-LATEX

Back in the introduction we said that AMS-LATEX was just a big macro package, the result of a marriage of LATEX and AMS-TEX designed to endow the powerful general-purpose LATEX package with the mathematical prowess of AMS-TEX without compromising the LATEX syntax. Most of that is true, except that AMS-LATEX is really just a document-style option (like `12pt`) that can be used within a "tweaked LATEX". The most visible part of this tweak is the *new font selection scheme of Mittelbach and Schöpf*, discussed more fully in the next chapter. Almost every LATEX command and environment survived the transition to AMS-LATEX, the exceptions being those that were considered redundant or under-used (space is at a premium with such a big package). The tweaked LATEX package is therefore able to deal with practically every existing LATEX document, giving just a few (often pleasant) surprises.

With the `amstex` style option, one can just start a LATEX document with

```
\documentstyle[amstex]{article} % or report, book, etc
```

to gain access to the TEXnical excellence of the AMS technical staff. It is not necessary to have read *The Joy of TEX* (the AMS-TEX reference guide) to be able to use the `amstex` option, for AMS-LATEX comes with its own reference guide. Even so, *The Joy of TeX* is still highly recommended reading. The syntax of AMS-TEX has been changed to that of LATEX, but one can perform that transformation as you read "*Joy*" and still learn much of the art of technical typesetting. Nowhere else will you find so comprehensive a coverage of the conventions and pitfalls of mathematical typesetting. In addition, *Joy* lists all the extra symbols that are available through the `amstex` option (if you thought LaTeX had a fair selection of esoteric symbols, just wait 'til you see those!) and provides in-depth accounts where the $\mathcal{AMS}$-LaTeX documentation is brief.

Just as $\mathcal{AMS}$-TeX comes with the AMS preprint style (`amsppt`), $\mathcal{AMS}$-LaTeX comes with a specialist style file for preparation of articles with $\mathcal{AMS}$-LaTeX for submission to journals: `amsart`. The $\mathcal{AMS}$-LaTeX User's Guide is quite short and very terse in its explanations (assuming you to be competent in LaTeX), but is supplemented by a large body of examples and a comprehensive sample article that is a showcase of the abilities of the `amsart` style. You must read both these documents to really learn $\mathcal{AMS}$-LaTeX. The AMS also distributes a guide to authors who wish to submit using $\mathcal{AMS}$-LaTeX, and this is a must-read once you are familiar with some of $\mathcal{AMS}$-LaTeX.