AN INTRODUCTION TO MATHEMATICAL DOCUMENT PRODUCTION USING $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$-LaTeX TEXNICCENTER EDITION

SIMON EVESON
EDITED BY TONY SUDBERY

Contents

Date: November 1994–July 1997.

Part 1. **Introduction to the TeX Family**

1. Introduction

These notes provide an *introduction* to mathematical document production on the York computer network using LaTeX 2$_\varepsilon$ (in particular $\mathcal{AMS}$LaTeX), BibTeX and `makeindex`. Although I have tried to cover the most important topics, there are many things I have left out. Much more information is available in the literature; for example:

$\mathcal{AMS}$LaTeX is thoroughly described in Grätzer's excellent book on the subject [3] (a pint to the first person to find *both* references to the University of York Mathematics Department). LaTeX 2$_\varepsilon$, BibTeX and `makeindex` are (reasonably) fully documented in Lamport's official user guide [5]; the earlier edition [6] is still useful. A much more detailed description (not for the faint-hearted) can be found in [1], which also includes a survey (inevitably, slightly out of date) of packages providing additional features. The knotty subject of including graphics within your document is thoroughly untangled in [2]. Finally, the underlying TeX engine that does the typesetting work is documented in [4].

I have made available online (at `http://maths.york.ac.uk/www/TexLaTeXInfo` the first chapter and the first two appendices (tables of symbols) of the first edition of Grätzer [3] and a cross-referenced file describing most common LaTeX 2$_\varepsilon$ commands.

2. The LaTeX Philosophy

LaTeX is not a word processor, but a document markup language. In working with LaTeX, you concentrate on the *structure* of your document, not the form. Working with a word processor, if you wanted a new section heading you would probably centre or flush the title, which you would type in a larger or bolder font, and you would type in the section number yourself. In LaTeX, you simply ask the system for a section heading. The system is responsible for the appearance of the heading, and for the generation of the section number. This ensures that the author is not troubled with irrelevant details, but can concentrate on the contents of the document, rather than its appearance. It ensures that section headings have a consistent appearance within the document, and allows this appearance to be changed with one modification to the document description. Automatic numbering allows sections to be added, removed and rearranged without worrying about the numbering scheme. Cross-references to sections and pages are done by labels rather than literal numbers, so they are always correct and never need to be manually updated.

Section headings are just one example of the way a LaTeX document is built. The same applies to displayed and numbered equations, theorems and enumerated lists.

Underlying the structured LaTeX interface is the typesetting engine TeX. TeX's ability to typeset mathematical formulae is unparalleled, and LaTeX and particularly AMSLaTeX give the same structured approach to building complex formulae as they do to building complex documents.

Working with LaTeX is very different to working with a WYSIWYG word processor. You do not see the final appearance of the document as you type it, but the control sequences describing its structure. When you want to preview or print it, you run it through LaTeX then through a previewing program or a printer driver. This gives something akin to the edit-compile-execute cycle associated with compiled computer languages.

Part 2. **Working with TeXnicCenter**

3. Setting up TeXnicCenter

TeXnicCenter is a *front end* for LaTeX i.e. a word processor which makes it easier to produce a LaTeX source file and to process it. It contains various menus which enable you to insert LaTeX commands at the click of a mouse, without having to remember the typed form of the commands; it also provides checks on the soundness of your LaTeX file. The first output (before processing) is a file with the extension .tex; this is a text file which you could have typed directly from the keyboard, and which you can edit in this way.

To use TeXnicCenter on your own machine, you will need to install MiKTeX and then download TeXnicCenter. Instructions for doing this are available at http://maths.york.ac.uk/www/UgradComputingDownloads.

To use TeXnicCenter on a Windows XP machine in a computer classroom at the University of York, open TeXnicCenter as follows: Click on the Start menu, then move the mouse pointer to Programs, then to Text Processing and click on TeXnicCenter.

Before using TeXnicCenter for the first time, you need to configure it. To do this, click the Build menu, then Define output profiles... Click on Wizard at the bottom left-hand corner of the window that comes up. In the new window, click on Next; then type `C:\tex\miktex\bin` in the box. Click on Next. The next window should have a button labelled “Finish” (if not, keep clicking “Next” until you reach this last window). Click on Finish; a series of dialogue boxes appears, telling you about output profiles and asking if they should be overwritten. Click on Yes every time, and finally OK.

A window headed “Profiles” remains. On the left-hand side of this window, click on LaTex=>DVI so that it is highlighted. Click on the tab Viewer, and type `C:\tex\miktex\bin\yap.exe` in the top box.

Finally, click OK.

It is very useful to have the lines of your tex file numbered on the screen. To get this to happen, click on the menu tools and then Options. In the window that comes up, click on the tab Text Format and tick the box labelled ”Show line numbers”.

Fixing LaTeX. If running LaTeX produces an error message ”Cannot find default format files”, or if it seems to be running very slowly, open a Web browser, go to `www-users.york.ac.uk/~spe1/texfix.bat`, and follow the instructions.

4. Creating a Simple LaTeX Document

A LaTeX document is a file (or collection of files) in plain, human-readable text, containing instructions to the TeX typesetting system. The following example is about the simplest possible LaTeX document:

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

Hello, world!

\end{document}
```

The first thing to do is to give the document a name. Click on the File menu then on Save As.... When a dialog box appears, navigate to the folder where you want to keep material related to this module, click in the box labelled "File name", type in `test.tex` (or any other name you'd like to use, as long as it ends in `.tex`) and either click on Save or press $\boxed{\text{Return}}$.

Now just type in the example as it is printed here. The line `Hello, world!` is the content of your document, and you can replace it by anything you like (within reason, and within the law), but the other three lines must be present in any LaTeX document (with possible variations in the first line, which we will discuss later).

You have now created a LaTeX document. To see it on the screen or print it out, you need to go through two more stages: first, you need to run it through LaTeX; then you need to either preview it or print it.

Before running the document through LaTeX, check that the dialogue box in the middle of the toolbar shows "LaTeX $\Rightarrow$ DVI". There are now three ways to run LaTeX. The first is to click on Build then on Current File. The second is to click on the appropriate icon on the toolbar (an icon can be identified by letting the arrow rest on it for a second: an explanation of the icon will appear just below it). The third way, using the keyboard, is to press $\boxed{\text{CTRL}}$+$\boxed{\text{F7}}$ (two keys simultaneously). Any of these will cause a window to open at the bottom of the screen, showing various messages as LaTeX proceeds through the document. These will include the lines

```
  I found no \citation commands---while reading file \ldots\test.aux
  I found no \bibdata commands---while reading file \ldots\test.aux
  I found no \bibstyle commands---while reading file \ldots\test.aux
  (There were 3 error messages)
  Couldn't find input index file M:\plain nor M:\plain.idx.
```

(if `test.tex` is the name of your document). Don't worry about these. The important lines are

```
Output written on test.dvi (1 page, xxx  bytes).
```

and

```
LaTeX-Result: 0 Error(s), 0 Warning(s), 0 Bad Box(es), 1 Page(s)
```

which show that you have successfully processed your document. If, instead of a clean stop, you see an error message beginning with an exclamation mark in the bottom screen, with a white-on-red cross in the margin, then you have made a typing error serious enough to stop LaTeX in its tracks (which is unlikely in a document as simple as this one). Get rid of the window by clicking on the cross in the top left-hand corner.

If running LaTeX produces an error message "Cannot find default format files", or if it seems to be running very slowly, open a Web browser and go to `www-users.york.ac.uk/~spe1/texfix.bat`, and follow the instructions.

If LaTeX runs successfully, it produces a file `test.dvi`, which can be used to view and print the document. There are three ways to preview the document. The first is to click on Build, then on View Output. The second is to click on the appropriate icon in the toolbar (next but one to the icon for running LaTeX, on its right). The third is to press the key $\boxed{\text{F5}}$. Any of these will produce a window labelled "Yap 0.99i - [test.dvi]" showing the typeset form of your document.

If you are happy with this form of the document, you can print it from this window by clicking on the printer icon in the toolbar (or by clicking on File, then Print). But when you see it, you may realise there is something you want to change or add. In that case, minimise the preview window and return to your document window in TeXnicCenter. Edit your file `test.tex` and run LaTeX again. If it runs successfully, there is no need to press View Output again: you can go straight back to the Yap window, which will already be showing the updated version of your document.

To help in this process of returning from the preview window to your LaTeX source file, the Yap preview of the document contains a sign (looking like an O clumsily written with a thick old paintbrush) at the point in the document corresponding to the position of the cursor in the TeXnicCenter window containing your source file.

Instead of `text.dvi`, you can get output in the form `test.ps` or `test.pdf`, which are more portable; in particular, `test.pdf` can be read by any computer. These are obtained by changing the box in the middle of the toolbar from "LaTeX $\Rightarrow$ DVI" to "LaTeX $\Rightarrow$ PS" or "LaTeX $\Rightarrow$ PDF", and then proceeding as above.

Part 3. **Ordinary Text**

5. The Structure of a LaTeX document

All LaTeX documents have the same general structure:

`\documentclass`{} declaration

(`\usepackage{}` declaration)
(preamble)
`\begin{document}`
(topmatter)
document body
`\end{document}`

where items in parentheses (i.e, round brackets) are optional.

The document class controls the broad appearance of the document, including for example the way section headings appear and the material that appears in page headers and footers. Different classes are used for different purposes. There are five classes present in standard LaTeX, and three more with AMSLaTeX. These are:

`article`: LaTeX article class, for short documents.
`book`: LaTeX book class.
`letter`: LaTeX letter class.
`report`: LaTeX report class, for longer documents than `article` class.
`slides`: LaTeX slides class, for OHP transparencies.
`amsart`: AMS article class, for mathematical preprints.
`amsbook`: AMS book class, for mathematical monographs.
`amsproc`: AMS proceeding class, for conference proceedings.

This is by no means an exhaustive list of possible document classes, but it covers the most common ones. Since these notes are about AMSLaTeX, we shall use only the two AMS classes, `amsart` for articles or preprints and `amsbook` for monographs, theses or books. The differences between the two classes are mostly cosmetic: for example, in book class the title page, table of contents and bibilography start on a fresh page, whereas in article class they do not. The only major non-cosmetic distinction is that in the book style the top-level sectioning unit is the chapter, whereas in the article style it is the section (more about sectioning units in Section 27).

Some fine details of the way the document class operates are controlled by the *class options*: here are the most common ones (this list is a long way from being exhaustive)

`11pt`: 11-point type; 10% larger than the default size.
`12pt`: 12-point type; 20% larger than the default size.
`a4paper`: A4 paper size; always use this option.
`onecolumn`: Print in one column the whole width of the page. This is the default in all classes described here.
`twocolumn`: Print in two columns half the width of the page.
`fleqn`: Align displayed equations on the left margin instead of centring them.
`leqno`: Print equation numbers on the left margin. This is the default in AMS classes.
`reqno`: Print equation number on the right margin. This is the default in standard LaTeX classes.
`oneside`: Print on one side of the paper. This is the default in the standard LaTeX classes.
`twoside`: Print on both sides of the paper. This is the default in the AMS classes. Note that this does not mean that the printer actually prints on both sides, just that the document is laid out for double-sided printing. To get a physically double-sided document you need to use an appropriate printer (or a printer and a photocopier).
`titlepage`: Put the title page and abstract on separate pages. This is the default in all classes other than `article` and `amsart`.

The options are given as a comma-separated list, placed in square brackets after the word `\documentclass` but before the name of the class. For example, to work in article class in 12-point type on A4 paper, type

```
\documentclass[12pt,a4paper]{article}
```

Finally, there are the *packages*: extra features to add to the system. Here is another partial list:

`a4`: narrower margins.
`a4wide`: even narrower margins.
`amsmath`: AMSLaTeX; unnecessary in AMS classes.
`babel`: non-English language support.
`color`: multi-coloured text.
`amsfonts`: AMS fonts; unnecessary in AMS classes.
`graphics`: including graphics generated by other programs.
`latexsym`: some extra symbols.

makeidx: support for `makeindex` indexing system.
pict2e: enhanced picture and diagram system.
showidx: prints index entries in the margin.
times: use PostScript fonts for running text.
mathptm: use PostScript fonts for mathematics.
mathrsfs: script capital letters in math mode.
amssymb: lots of extra mathematical symbols.
verbatim: enhanced `verbatim` and `comment` environments.
diagrams: Paul Taylor commutative diagram package.
amscd: $\mathcal{AMS}$ commutative diagram package (use `diagrams` in preference).
autograph: a graphics package for automata.
pstricks: a general graphics package.

Packages are loaded with the `\usepackage{}` command. The list of packages is enclosed in braces, separated by commas. For example, to load the two Times Roman packages type

```
\usepackage{times,mathptm}
```

You can have as many `\usepackage{}` declarations as you want, each of which can contain as many packages as you want. An alternative way of entering the last example is:

```
\usepackage{times}
\usepackage{mathptm}
```

For most packages, the order in which you list them is irrelevant. However, the `pstricks` package must come before `a4` or `a4wide`.

Once the document skeleton has been created, there are two points where you type more instructions: the *preamble* and the *document body*. The preamble is between the `\usepackage{}` declaration and the `\begin{document}` line; the document body is between this and the `\end{document}` lines. Anything after `\end{document}` is ignored. Some of the uses of the preamble will emerge later, but we shall now move on to the document body, which is where the text of document lives.

6. Lines and Paragraphs

Most ordinary text can be typed exactly as it appears. The essential point to remember is that line breaks and spaces in your input file do not correspond to line breaks and spaces in the final output. For example, in this paragraph the line breaks thus far in the source file are after ‘essential’, ‘do’, ‘For’ and ‘file’. Paragraph breaks are indicated by a blank line in the source file (press $\boxed{\text{Return}}$ twice).

When TEX is in paragraph mode, it reads through a paragraph of text, breaking it down into words which can be separated by any amount of *white space*: spaces and line breaks. The number of spaces separating words and the position of line breaks in the source file are entirely ignored. Each word is typeset and added to the end of the line being formed until the line overflows the right margin. At this point, TEX will either squash the text on the line together to accommodate the last word, move the last word on to the next line and stretch the current line to meet the margins, or combine one of these actions with hyphenating the word. When it meets a blank line in the input it moves onto the next output line and indents, ready for the next paragraph to start. Some examples:

```
This is one way of typing a very dull example paragraph whose only
virtue is that it will be typeset over more than one line.

One dull example deserves another.
```

This is one way of typing a very dull example paragraph whose only virtue is that it will be typeset over more than one line.

One dull example deserves another.

```
    This is another way of            typing a very
dull
 example
 paragraph
whose only virtue is that it too will be typeset over more than
one
```

```
line.

One dull example
                                             deserves
                                                                 another.
```

This is another way of typing a very dull example paragraph whose only virtue is that it too will be typeset over more than one line.

One dull example deserves another.

As you can see, spaces and linebreaks are entirely interchangeable, the only special signal being an entirely blank line which starts a new paragraph.

Punctuation characters are typed as they appear on the keyboard, and should be followed by at least one space or carriage return. There is no need to put more than one space: any number of spaces is equivalent to one. TeX adjusts the amount of space after the punctuation automatically, giving more space after sentence-ending punctuation like full stops or exclamation marks.

Abbreviations like e.g. or i.e. need special treatment, because they fool TeX into thinking that the sentence has ended. To stop it adding extra space, you should place a *hard space* or *control space* after the abbreviation, which is a backslash `\` followed by a space. The first line of this paragraph was typed as

```
Abbreviations like e.g.\ or i.e.\ need special ...
```

The first line of a new paragraph is normally indented. You can stop the automatic indentation at the beginning of a paragraph by putting the instruction `\noindent` after the paragraph break. Note the backslash before noindent: this is what tells TeX that it is an instruction to the computer, not a word to be typeset.
For example, this paragraph beings with `\noindent`. Similarly, you can force the beginning of a paragraph to be indented by putting `\indent` at the beginning of it (this is very rarely necessary: see Section 27 for the most common use of `\indent`). `\indent` and `\noindent` are called *control sequences* or *macros*; see Section 9 for a brief description of what a macro is.

7. Quotation Marks

In TeX, quotation marks are handled in what may seem at first sight to be an unusual way. The two symbols used are `'` (apostrophe: centre right of the keyboard) and `` ` `` (back-quote: top left of the keyboard). Single quotation marks are generated using a back-quote as the left mark and an apostrophe as the right mark, and double quotation marks are handled by typing the quote marks twice, like this:

```
`single quotes' `look like' `this';
``double quotes'' ``look like'' ``this''.
```

‘single quotes’ ‘look like’ ‘this’; “double quotes” “look like” “this”.

The `"` symbol is not allowed in a .tex file. The reason this scheme is used is simply that standard computer keyboards have only one double-quote symbol, where two are actually needed. However, TeXnicCenter, like some non-technical word-processors, allows you to use the key $\boxed{"}$. If you type $\boxed{"}$ after a space, TeXnicCenter will insert ``` `` ``` into your .tex file; otherwise it will insert `''`. So typing $\boxed{"}$`this`$\boxed{"}$ produces ``` ``this'' ``` in your .tex file and “this” in your typeset output. You can remove this feature (as you will have to if you want to produce an umlaut – see Section 10) by clicking on Tools, then Options, and removing the tick from the box “Replace quotation marks” under the “General” tab.

8. Dashes

There are three kinds of dash symbol available in TeX: hyphens, en-dashes and em-dashes, generated by `-`, `--` and `---`. A hyphen `-` is used for hypenated words, such as X-ray, an en-dash `--` is used for numeric ranges such as 1–10, and an em-dash `---` is used for punctuation—like this. (Note that it is common practice not to type spaces before or after an em-dash. However, this is a matter of taste.)

9. Special Symbols

While most plain text can be typed as it appears, there are a number of symbols which have a special meaning to TeX, or which for technical reasons cannot be typed in the usual way. These are

$ % & ˜ _ ^ { } | < > \

To type these symbols, we need to use *control sequences*. A control sequence (or *macro*) is a word in your input file which is not interpreted as text to be typeset as it appears, but is instead an instruction to TeX to take some particular action. The simplest of these just instruct TeX to print some particular symbol which cannot be typed directly. Almost all control sequences consist of a `\` character followed by either of a word or a single special symbol (which is why \ cannot be produced by typing `\`). Here are some macros for generating textual symbols:

Text-Generating Macros											
`\#`	#	`\$`	$	`\%`	%	`\&`	&	`\_`	_	`\{`	{
`\}`	}	`\oe`	œ	`\OE`	Œ	`\ae`	æ	`\AE`	Æ	`\aa`	å
`\AA`	Å	`\o`	ø	`\O`	Ø	`\l`	ł	`\L`	Ł	`\ss`	ß
`?‘`	¿	`!‘`	¡	`\dag`	†	`\ddag`	‡	`\S`	§	`\P`	¶
`\copyright`	©	`\pounds`	£	`\TeX`	TeX	`\LaTeX`	LaTeX	`\AmS`	AMS	`\ldots`	...

The ellipsis . . . can always be generated by `\ldots`, but in AMS classes can also be generated by `\dots`. This doesn't really matter when typing text, but it is rather more significant when typing mathematics: see Section 19 for details.

In addition, `\today` generates today's date, in this case October 17, 2007.

This does not entirely solve the problem of how to type every character on the keyboard: the symbols `<`, `|`, `>`, `\`, ˜ and ˆ are still missing. The first four are available as math-mode symbols (see Section 18), the last two are not. [1]

Almost all of these names begin with a backslash character (the only two which do not are `?‘` and `!‘`, which we shall ignore for the time being[2]). Those that do begin with a backslash, the vast majority, can be divided into two classes: *control words* and *control symbols*. A control word begins with a backslash and is followed by one or more letters, such as `\AE` or `\TeX`. A control symbol consists of a backslash followed by exactly one symbol, which is not a letter, such as `\#` or `\$`.

When you type a control word, you need to be careful about what follows it. For example, to produce Æsop you can't type `\AEsop` because TeX tries to interpret a command called `\AEsop`, not the command `\AE` followed by the letters sop. The two common ways to make TeX realise that you want the macro called `\AE` is to place a null brace pair or a space after it: Æsop can be produced by typing `\AE{}sop` or `\AE sop`. In the second form, the space is used only to tell TeX that you have finished the name of the macro you want, it is not produced in the text. This is the correct behaviour in this case, but sometimes you do want a space after the macro's text. For example, to produce 'TeX commands' you can't type `\TeX commands` because the space won't be produced. The two standard solutions here are to use a null brace pair as well as a space (`\TeX{} commands`) or to use a *hard space* (`\TeX\ commands`). See Section 6 for the other standard use of hard spaces.

10. Accents

There are also macros for generating accented letters. These are slightly more complicated than the symbol-generating macros described above, in that they take an *argument*. An argument to a macro is a piece of text supplied to the macro for it to process—in this case, simply the letter on which you want the accent. Arguments are typed after the macro name and are normally enclosed in braces (this is why the { and } characters have to be typed as `\{` and `\}`, not just as { and }).

Textual Accents											
`\‘{a}`	à	`\’{a}`	á	`\^{a}`	â	`\"{a}`	ä	`\~{a}`	ã	`\={a}`	ā
`\.{a}`	ȧ	`\u{a}`	ă	`\v{a}`	ǎ	`\H{a}`	a̋	`\t{aa}`	a͡a	`\c{a}`	a̧
`\d{a}`	ạ	`\b{a}`	a̱								

Note that the letters i and j should not be accented as they are; they should lose their dots. Dotless i and j characters ı and ▮ are obtained with `\i` and `\j`, so í is produced by `\'\i`. You will have noticed the slug ▮ above in place of a dotless j. This is because this document is set in Times Roman using the `times` package, which generally gives a better appearance, but lacks the dotless j glyph. If you actually need this character, you will need to set your document in the default Computer Modern fonts (that is, do not put `\usepackage{times}` in your preamble).

[1]For the record, ˜ and ˆ can be obtained by typing `\char‘\~` and `\char‘\^`.

[2]OK, I admit it, they're not macros at all: they're ligatures.

11. Literal Space

If you want to add a bit more space at some point, the macros `\quad` and `\qquad` are often useful: `\quad` inserts a block of horizontal space equal to 1 em (the width of a capital M in the current font), and `\qquad` inserts twice this much space. Smaller amounts of horizontal space are inserted by `\,` or `\:` or `\;` (in ascending order of size); these are used mainly for fine adjustments in mathematical formulae.

For vertical space, there are macros `\smallskip`, `\medskip` and `\bigskip`. The exact height of a `\smallskip` depends on what class you are using, but a `\medskip` is worth two `\smallskip`s and a `\bigskip` is worth two `\medskip`s. For example:

```
hello\quad goodbye\qquad hello again

\bigskip
goodbye again
```

hello goodbye   hello again

goodbye again

Notice that there is a paragraph break before the `\bigskip`. The three skip commands are only effective between paragraphs (in what TEX calls *vertical mode*). If you use a skip command inside a paragraph, it will have no effect.

There are a few occasions when you want to force TEX to insert a block of literal space of a given length or height. Try to avoid these commands as far as possible — there are almost always better ways of doing things, for example the commands above. However, if you really must, the macros `\hspace`{} and `\hspace*`{} insert horizontal space and the macros `\vspace`{} and `\vspace*`{} insert vertical space. They all take one argument, a length which can be specified in various units, such as `mm`, `cm` or `in`. For example, `\hspace{25mm}`, `\hspace{2.5cm}` and `\hspace{1in}` all give about the same amount of horizontal space.

The difference between `\hspace`{} and `\hspace*`{} is that the space given by `\hspace*`{} is always added, even if it falls at the end of a line, whereas that added by `\hspace`{} is suppressed if it falls at the end of a line. Similarly, `\vspace*`{} produces vertical space even at the end of a page, whereas the space produced by `\vspace`{} is suppressed if it falls between pages. Like the skip commands, space produced by `\vspace`{} is only visible if it is used between paragraphs.

The main problem with specifying literal space in terms of absolute units like `mm` or `in` is that these units take no notice of global size changes. If you design a document in ten-point type and then decide it will look better in larger twelve-point type, you can simply change the class options to

```
\documentclass[12pt]{amsart}
```

However, any literal spaces in your document will not change size if they are specified in fixed units. This might merely look a bit odd, or it might completely ruin the layout. A better solution is to use units called `em` (for horizontal space) and `ex` (for vertical space). One *em* is the width of a capital M in the current font; one *ex* is the height of a lowercase x in the current font. If you specify all your lengths in terms of these units, then they will scale with the document.

12. Controlling Hyphenation, Line and Page Breaks

Although TEX generally breaks lines and pages at good locations, there are times when things go wrong. The first precaution to take is to use *ties*. A tie is typed ~ and appears as a space in the final outcome, but the line is never broken at a tie. There are certain cases when you should almost always use ties instead of spaces, particularly in names: a name should never be broken between its initials, so you should type C. F. Gauss as `C.~F.~Gauss`. Similarly, expressions such as 'no. 3', 'Section 2' or 'YO10 5DD' should be typed with ties (`no.~3`, `Section~2`, `YO10~5DD`) to prevent bad line breaking.

You can stop TEX hyphenating words by using the `\mbox` macro, which typsets its argument in an object known as an *LR box*. The details of what LR boxes do are beyond the scope of these notes, but for now all you need to know is that they never break across lines and spaces inside them are not stretched or compressed when TEX is making a flush right margin.

For example, typing

```
\mbox{this is an excessively long LR box which will probably
  cause no end of trouble because \TeX\ will be unable to
  break it anywhere and it might even run off the page}
```

has the following effect:

this is an excessively long LR box which will probably cause no end of trouble because TEX will be unable to break it anywhere and it r

which illustrates the restraint that should be employed when using `\mbox` to suppress line breaks.

TEX is very good at hyphenation, but not perfect. If you want to allow it to hyphenate a word in a place which its algorithm does not allow, you can use a *discretionary hyphen*, which is indicated by `\-`. For example, TEX is unable to hyphenate the word *galaxy*. To tell it that it is acceptable to hyphenate it as *gal-axy* if required, type it as `gal\-axy`. The discretionary hyphen will not appear unless TEX needs to break the line in the middle of the word.

Alternatively, you can use the `\hyphenate{}` command once in the preamble: once TEX has seen the command `\hyphenate{gal-axy}`, it will apply the hyphenation whenever it is required, without the use of discretionary hyphens.

You might very occasionally need to put an explicit line break into a paragraph. The command for this is unsurprisingly named `\linebreak`. For example,

```
Stop in the middle of a \linebreak line and start on the next line.
```

Stop in the middle of a
line and start on the next line. There is a corresponding command `\nolinebreak` which prohibits a line break at the point where it appears.

As you can see, the line before the break is spread out to fill the page. There is a similar command which does not do this: `\newline`. For example,

```
Stop in the middle of a \newline line and start on the next line.
```

Stop in the middle of a
line and start on the next line.

To handle awkward page breaks, we have `\pagebreak` and `\nopagebreak`. These are a little more complicated than `\linebreak` and `\nolinebreak`: their action depends on their location in the document. If they are between paragraphs, they act in the obvious way, causing or inhibiting a page break at that point. If they are inside a paragraph, they apply to the end of the current line. Thus, in paragraph mode, `\pagebreak` means 'break the page when you get to the end of this line' and `\nopagebreak` means 'don't break the page when you get to the end of this line.' If you really want to break a page in the middle of the current paragraph, use `\linebreak\pagebreak`.

If you have a block of material which you want to keep together on one page but which slightly overflows, you can use `\enlargethispage{}` to add a little more space to the current page. This macro takes one argument, a length. Like the argument to `\vspace{}`, it is probably best to specify this in terms of `ex`: something like `\enlargethispage{3ex}` should squeeze in one more line.

Finally, we mention `\clearpage` and `\cleardoublepage`. You won't fully understand what they do until you've read Section 35, but `\clearpage` first processes any pending floats (this is what you need to read Section 35 for!), and then acts like `\pagebreak`.

In a single-sided document class (`article`, `report`, or any class declared with the option `oneside`), the command `\cleardoublepage` acts exactly like `\clearpage`.

In a double-sided class (any $\mathcal{AMS}$ class, `book`, or any class declared with the option `twoside`), the action of `\cleardoublepage` is first to act like `\clearpage` and then to check that the new page has an odd page number. If it does not, another `\pagebreak` is executed. This guarantees that the text following a `\cleardoublepage` always starts on a right-hand (odd-numbered) page. It is most commonly used when you are starting a new section in the document, which you want to start at the top of a right-hand page.

13. Changing Font Style

There are three basic ways of changing the LATEX font style: by macro, declaration and environment. The first way, by macro, is normally used if you want to typeset one or two words in a different font or style. Like the accenting macros in Section 10, font-changing macros take one argument, which follows the macro name enclosed in braces. For example, to typeset the word 'hello' in italic like this: *hello*, type `\textit{hello}`.

There are a total of ten short font-changing commands.

Short Font-Changing Macros							
button	macro	meaning	example	button	macro	meaning	example
H!	`\emph{}`	emphasise	*example*	**F**	`\textbf{}`	bold	**example**
K	`\textit{}`	italic	*example*	*S*	`\textsl{}`	slanted	*example*
`T`	`\texttt{}`	typewriter	`example`	KA	`\textsc{}`	small capitals	EXAMPLE

In TeXnicCenter, some font-changing commands are available from the toolbar, by clicking on the buttons whose labels are shown in the above table. (These buttons can be found in fourth row of the TeXnicCenter window.)

The last two, `\texmd{}` and `\textup{}`, are sufficiently obscure that you will probably never need them, because upright medium-weight letters are the default. `\textrm{}` is also uncommon, because Roman (serifed) letters are the default.

These macros can be combined in various ways to produce more complicated font styles:

```
\textbf{\textit{bold italic}}
\textbf{\texttt{\textsl{bold slanted typewriter}}}
```

bold italic ***`bold slanted typewriter`***

The `\emph{}` macro deserves special mention. It is used to emphasise text, and does not always produce the same font. In ordinary text, `\emph{}` normally switches the font to italic; however, if the running text is currently italic, then `\emph{}` switches to Roman. This ensures that emphasised text stands out, regardless of the style of the running text. This is particularly useful because in some document classes certain portions of text (such as theorem statements) are automatically italicised, and in others they are not. Using `\emph{}` instead of `\textit{}` ensures that there is emphasis in either kind of class.

The font-changing macros above are normally used when you want to change one or two words into a different font. If you want to typeset an entire paragraph in a different font, the *declaration* style of font-changing command is often more convenient.

Here is a complete list of the long font-changing commands.

Font-Changing Declarations					
macro	meaning	macro	meaning	macro	meaning
`\bfseries`	bold	`\scshape`	small capitals	`\em`	emphasise
`\itshape`	italic	`\slshape`	slanted	`\mdseries`	medium weight
`\rmfamily`	roman	`\sffamily`	sans serif	`\ttfamily`	typewriter
`\upshape`	upright				

A font-changing declaration such as `\itshape` does not take an argument, but changes the font to italic until further notice. Some brief TEXnicalities: all declarations have a *scope*, the region of the document they affect, which for font-changing declarations is until the end of the current *group*. A group is a unit of text between a pair of delimiters, normally either a brace pair or a `\begin{}` ... `\end{}` pair (the only one of these you have seen so far is `\begin{document}` ... `\end{document}`).

What this boils down to is that font-changing declarations are normally used like this:

```
{\bfseries this is a longer unit of text, set in bold}
```

this is a longer unit of text, set in bold

The left brace opens a new group, the `\bfseries` declaration changes the font to bold for the rest of the group, and the right brace closes the group, which resets the font to its original value.

14. CHANGING SIZE

The size of the current font may be changed using the following declarations, which are used in the same way as the font-changing declarations above:

Size-Changing Declarations							
macro	example	macro	example	macro	example	macro	example
`\Tiny`	example	`\tiny`	example	`\SMALL`	example	`\Small`	example
`\small`	example	`\normalsize`	example	`\large`	example	`\Large`	example
`\LARGE`	example	`\huge`	example	`\Huge`	example		

For example,

```
{\small small is} {\Huge beautiful}
```

small is beautiful

Size-changing should be treated with care. Inappropriate use will result in your document being uncomfortable to read. Note that `\Tiny`, `\Small` and `\SMALL` are available only in the $\mathcal{AMS}$ document classes.

There are two further size-changing commands, which respectively give the size used for footnotes and for subscripts or superscripts in the current class:

Further Size-Changing Declarations			
macro	example	macro	example
`\scriptsize`	example	`\footnotesize`	example

Finally, in the $\mathcal{AMS}$ document classes there are two relative size-changing commands, which change to the next smaller or next larger size from the list above: `\smaller` and `\larger`.

15. Comments

If you put a `%` character in your text, everything from that point until the end of the line is ignored by TEX. This is useful for leaving reminders to yourself in the file, for example

```
%% Check reference for this result
```

Putting in two `%` characters is unnecessary, but it makes it more convenient to search through the document for any comments you left yourself.

Commenting out is a useful technique when you have some LATEX source which has errors which you cannot identify. To continue working on the rest of the document, put `%` signs at the front of the offending lines, so they are ignored by TEX. You can then fix them at your leisure.

An alternative way, useful for large blocks of text, is to use the `comment` environment. You first need to add `verbatim` to your `\usepackage{}` declaration (see Section 5). Having done this, you can comment out a region of text by putting `\begin{comment}` before it and `\end{comment}` after it. TEX completely ignores anything between these lines.

Environments are a very important concept in LATEX: an environment is a unit of text enclosed in a pair of delimiters `\begin{name}` and `\end{name}`, which is handled in a particular way; in this case, it is ignored completely. You will see many more examples of environments later.

In TeXnicCenter a large block of text can be commented out by highlighting it (e.g. by sweeping the cursor over it with the left mouse button pressed down) and then clicking on Edit, then Insert Block Comment. You can remove the comments from a block of text by selecting the block, clicking on Edit and then on Remove Block Comment (or Toogle (sic) Block Comment, which is kindly provided for people who have difficulty remembering what they decided to do two seconds ago).

Part 4. Math Modes

16. Entering and Leaving In-line and Displayed Math Mode

So far, you have seen two of LATEX's different modes of operation: paragraph mode and (briefly) LR mode. The third important mode is *math mode*, subdivided into *in-line* or *text-style* and *display-style* modes.

Within a paragraph, in-line mathematical formulae are enclosed between `$` signs, so $2 + 2 = 4$ is typed as `$2+2=4$`. An alternative method, which has some advantages but is not so common, is to use `\(` instead of the first `$` sign and `\)` instead of the second, so $2 + 2 = 4$ could also be typed as `\(2+2=4\)`. To display a formula, use square brackets instead, so that the formula is enclosed between `\{` and `\]`. Thus

$$2 + 2 = 4$$

is typed as

```
\[ 2+2=4 \]
```

(This also has an alternative, in which `\[` and `\]` are both replaced by `$$`.)

It is good practice to type a displayed formula on a line by itself, as this makes it stand out in your source file. Some authors go further, and format a display like this:

```
\[
  2+2=4
\]
```

to make it stand out even more. If you want punctuation at the end of the display, put it *inside* the display, like this:

```
\[ x(y+z)=xy+xz. \]
```

$$x(y+z) = xy + xz.$$

If the full stop came after the `\[`, it would appear on the next line. In contrast, if you want to put punctuation after in-line mathematics, it goes *after* the second `$`. Getting this right in in-line mathematics is less important than in displayed mathematics: if you get it wrong, the worst that can happen is that you end up with interword spacing instead of intersentence spacing at the end of a sentence.

In the output of math mode, letters appear in italic, spaces are ignored entirely except that they delimit macro names, and 'words' are printed as if they represent products of symbols, which are spaced entirely differently to ordinary English words. Thus, typing `$x=y iff y=x$` produces the output $x = y iff y = x$, which is unlikely to be what you want. To put ordinary words inside a mathematical formula, use the `\text{}` macro:

```
\[ x=y \text{ iff } y=x \]
```

$$x = y \text{ iff } y = x$$

Note that `\text{}` takes leading and trailing spaces literally, so `\text{ iff }` produces spaces on both sides of "iff".

Note also that `\text{}` is only available in the AMS classes; in other classes you can use `\mbox{}` instead, but be aware that while `\{}` changes the size of text to suit its position in the formula, `\mbox{}` gives the same size in all positions. In particular, it gives the wrong size in subscripts and superscripts. Both use whatever font style was in use when math mode was entered, rather than the document default. Thus, if a theorem statement is in italic, `\text{}` and `\mbox{}` make words inside math mode also italic rather than roman. If you find this sometimes undesirable (as I do, for example in subscripts and superscripts), use `\textup{}` instead of `\text{}`.

Several special characters, some of which do not work in paragraph mode, do what you would expect in math mode; in particular, $+$, $-$, $<$, $= >$, $|$, $/$ and . operate in a natural way.

17. Log-like Functions

One consequence of the way letters perform in math mode is that typing the names of standard functions as they appear produces incorrect output: typing `sin(x)` in math mode produces $sin(x)$; the sin is in italic, where it should be Roman, and the letter spacing is wrong.

There is a standard set of *log-like function macros* available to cope with this problem, as follows.

Log-Like Functions									
`\arccos`	arccos	`\arcsin`	arcsin	`\arctan`	arctan	`\arg`	arg	`\cos`	cos
`\cosh`	cosh	`\cot`	cot	`\coth`	coth	`\csc`	csc	`\deg`	deg
`\det`	det	`\dim`	dim	`\exp`	exp	`\gcd`	gcd	`\hom`	hom
`\ker`	ker	`\lg`	lg	`\ln`	ln	`\log`	log	`\Pr`	Pr
`\sec`	sec	`\sin`	sin	`\sinh`	sinh	`\tan`	tan	`\tanh`	tanh

So, to typeset $\log(x) = \cos(y)\sin(z)$, type `$\log(x)=\cos(y)\sin(z)$`.

Some other functions (*operators* like $\min$ and $\lim$, which take limits) are described in Section 20. You can also define your own log-like functions; see Section 39.

18. Mathematical Symbols

LaTeX and particularly AMSLaTeX have a very wide range of mathematical symbols associated with them. The following tables are reasonably exhaustive, and may look a little intimidating at first sight. Remember though that you will never need many of the symbols here, and that those you do need can be learned gradually. Because of the multiple sources of these symbols, there is no complete set of tables in any of the 'official' documentation but there is another more or less complete set (possibly more complete than mine) in Appendix A of Grätzer [3].

For historical reasons, these symbols fall into three classes. Some are always available, some are only available after the `amssymb` package has been loaded (by putting it into your `\usepackage{}` declaration: see Section 5), and a few are available after either `amssymb` or `latexsym` has been loaded. For practical purposes, you may as well just use the `amssymb` package whenever you need any of the more exotic symbols, and forget about `latexsym` altogether.

Greek letters can easily be obtained in math mode: the name of the macro is just the name of the letter, so `$\gamma$` produces γ. Upper-case letters have the first letter of the control sequence capitalised, so `$\Gamma$` produces Γ. Here is a complete list:

Greek Letters: standard LaTeX											
`\alpha`	α	`\beta`	β	`\chi`	χ	`\delta`	δ	`\epsilon`	ϵ	`\phi`	ϕ
`\gamma`	γ	`\theta`	θ	`\iota`	ι	`\kappa`	κ	`\lambda`	λ	`\mu`	μ
`\nu`	ν	`\omega`	ω	`\pi`	π	`\psi`	ψ	`\rho`	ρ	`\sigma`	σ
`\tau`	τ	`\upsilon`	υ	`\xi`	ξ	`\eta`	η	`\zeta`	ζ		
`\varepsilon`	ε	`\varphi`	φ	`\varpi`	ϖ	`\varrho`	ϱ	`\varsigma`	ς	`\vartheta`	ϑ
`\Delta`	Δ	`\Phi`	Φ	`\Gamma`	Γ	`\Theta`	Θ	`\Lambda`	Λ	`\Omega`	Ω
`\Pi`	Π	`\Psi`	Ψ	`\Sigma`	Σ	`\Upsilon`	Υ	`\Xi`	Ξ		

Greek letters: `amssymb` package			
`\digamma`	$\digamma$	`\varkappa`	$\varkappa$

There are a very few Hebrew letters available.

Hebrew Letters: standard LaTeX	
`\aleph`	$\aleph$

Hebrew Letters: `amssymb` package					
`\beth`	$\beth$	`\gimel`	$\gimel$	`\daleth`	$\daleth$

The ‘miscellaneous symbols’ below have nothing much in common, except that TeX treats them as if they were a letter or a digit, as opposed to something like a binary operator, which has more complicated spacing rules (TeXnically, these are math symbols of type ord).

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

Miscellaneous Symbols: `latexsym` or `amssymb` package					
`\mho`	$\mho$	`\Box`	$\Box$	`\Diamond`	$\Diamond$

Miscellaneous Symbols: `amssymb` package							
`\hslash`	$\hslash$	`\vartriangle`	$\vartriangle$	`\triangledown`	$\triangledown$	`\square`	$\square$
`\lozenge`	$\lozenge$	`\circledS`	$\circledS$	`\measuredangle`	$\measuredangle$	`\nexists`	$\nexists$
`\Finv`	$\Finv$	`\Game`	$\Game$	`\Bbbk`	$\Bbbk$	`\backprime`	$\backprime$
`\varnothing`	$\varnothing$	`\blacktriangle`	$\blacktriangle$	`\blacktriangledown`	$\blacktriangledown$	`\blacksquare`	$\blacksquare$
`\blacklozenge`	$\blacklozenge$	`\bigstar`	$\bigstar$	`\sphericalangle`	$\sphericalangle$	`\complement`	$\complement$
`\eth`	$\eth$	`\diagup`	$\diagup$	`\diagdown`	$\diagdown$		

The next block of symbols are called *binary operators*. They normally appear between two symbols, such as $A \cap B$ or $X \wedge Y$, and TeX ensures that the correct amoung of space appears between the operator and the symbols surrounding it. Some simpler characters, such as $+$ and $-$, are also binary operators.

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

Binary Operators: `latexsym` or `amssymb` packages							
`\lhd`	$\lhd$	`\rhd`	$\rhd$	`\unlhd`	$\unlhd$	`\unrhd`	$\unrhd$

Binary Operators: `amssymb` package							
`\dotplus`	$\dotplus$	`\smallsetminus`	$\smallsetminus$	`\Cap`	$\Cap$	`\doublecap`	$\doublecap$
`\Cup`	$\Cup$	`\doublecup`	$\doublecup$	`\barwedge`	$\barwedge$	`\veebar`	$\veebar$
`\doublebarwedge`	$\doublebarwedge$	`\boxminus`	$\boxminus$	`\boxtimes`	$\boxtimes$	`\boxdot`	$\boxdot$
`\boxplus`	$\boxplus$	`\divideontimes`	$\divideontimes$	`\ltimes`	$\ltimes$	`\rtimes`	$\rtimes$
`\leftthreetimes`	$\leftthreetimes$	`\rightthreetimes`	$\rightthreetimes$	`\curlywedge`	$\curlywedge$	`\curlyvee`	$\curlyvee$
`\circleddash`	$\circleddash$	`\circledast`	$\circledast$	`\circledcirc`	$\circledcirc$	`\centerdot`	$\centerdot$
`\intercal`	$\intercal$						

Binary relations are treated in a similar way to binary operators, with their own spacing rules.

Binary Relations: standard LaTeX									
`\leq`	$\leq$	`\prec`	$\prec$	`\preceq`	$\preceq$	`\ll`	$\ll$	`\subset`	$\subset$
`\sqsubseteq`	$\sqsubseteq$	`\in`	$\in$	`\vdash`	$\vdash$	`\geq`	$\geq$	`\succ`	$\succ$
`\succeq`	$\succeq$	`\gg`	$\gg$	`\supset`	$\supset$	`\supseteq`	$\supseteq$	`\sqsupseteq`	$\sqsupseteq$
`\ni`	$\ni$	`\dashv`	$\dashv$	`\equiv`	$\equiv$	`\sim`	$\sim$	`\simeq`	$\simeq$
`\asymp`	$\asymp$	`\approx`	$\approx$	`\cong`	$\cong$	`\doteq`	$\doteq$	`\propto`	$\propto$
`\models`	$\models$	`\perp`	$\perp$	`\mid`	$\mid$	`\parallel`	$\parallel$	`\bowtie`	$\bowtie$
`\smile`	$\smile$	`\frown`	$\frown$						

Binary Relations: `latexsym` or `amssymb` packages					
`\sqsubset`	$\sqsubset$	`\sqsupset`	$\sqsupset$	`\Join`	$\Join$

Binary Relations: `amssymb` package					
`\leqq`	$\leqq$	`\leqslant`	$\leqslant$	`\eqslantless`	$\eqslantless$
`\lesssim`	$\lesssim$	`\lessapprox`	$\lessapprox$	`\approxeq`	$\approxeq$
`\lessdot`	$\lessdot$	`\lll`	$\lll$	`\llless`	$\llless$
`\lessgtr`	$\lessgtr$	`\lesseqgtr`	$\lesseqgtr$	`\lesseqqgtr`	$\lesseqqgtr$
`\doteqdot`	$\doteqdot$	`\Doteq`	$\Doteq$	`\risingdotseq`	$\risingdotseq$
`\fallingdotseq`	$\fallingdotseq$	`\backsim`	$\backsim$	`\backsimeq`	$\backsimeq$
`\subseteqq`	$\subseteqq$	`\Subset`	$\Subset$	`\sqsubset`	$\sqsubset$
`\preccurlyeq`	$\preccurlyeq$	`\curlyeqprec`	$\curlyeqprec$	`\precsim`	$\precsim$
`\precapprox`	$\precapprox$	`\vartriangleleft`	$\vartriangleleft$	`\trianglelefteq`	$\trianglelefteq$
`\vDash`	$\vDash$	`\Vvdash`	$\Vvdash$	`\smallsmile`	$\smallsmile$
`\smallfrown`	$\smallfrown$	`\bumpeq`	$\bumpeq$	`\Bumpeq`	$\Bumpeq$
`\varpropto`	$\varpropto$	`\blacktriangleleft`	$\blacktriangleleft$	`\blacktriangleright`	$\blacktriangleright$
`\therefore`	$\therefore$	`\geqq`	$\geqq$	`\geqslant`	$\geqslant$
`\eqslantgtr`	$\eqslantgtr$	`\gtrsim`	$\gtrsim$	`\gtrapprox`	$\gtrapprox$
`\gtrdot`	$\gtrdot$	`\ggg`	$\ggg$	`\gggtr`	$\gggtr$
`\gtrless`	$\gtrless$	`\gtreqless`	$\gtreqless$	`\gtreqqless`	$\gtreqqless$
`\eqcirc`	$\eqcirc$	`\circeq`	$\circeq$	`\because`	$\because$
`\thicksim`	$\thicksim$	`\thickapprox`	$\thickapprox$	`\supseteqq`	$\supseteqq$
`\Supset`	$\Supset$	`\sqsupset`	$\sqsupset$	`\succcurlyeq`	$\succcurlyeq$
`\curlyeqsucc`	$\curlyeqsucc$	`\succsim`	$\succsim$	`\succapprox`	$\succapprox$
`\vartriangleright`	$\vartriangleright$	`\trianglerighteq`	$\trianglerighteq$	`\Vdash`	$\Vdash$
`\shortmid`	$\shortmid$	`\shortparallel`	$\shortparallel$	`\between`	$\between$
`\pitchfork`	$\pitchfork$	`\backepsilon`	$\backepsilon$		

Any math-mode symbol may be 'negated' by preceding it with the `\not` macro. This operates by drawing a stroke through the next symbol, so `\not=` produces $\neq$. However, `\not` does not know anything about the symbol it is negating, it just draws the same size of stroke. This can lead to the stroke dominating the symbol, as in the horrible $\not\sim$ which was produced by `\not\sim`. There are therefore some negated relations to cope with this case, all part of the `amssymb` package except `\neq` which is actually just a shorthand for `\not=`. For example, $\nsim$ is produced by `\nsim`, a great improvement over $\not\sim$. Although you can use `\not` to generate some of these, the symbols below give noticeably better results. Some cannot be generated by `\not` at all, as only part of the relation is negated, for example $\subsetneq$ which is given by `\subsetneq`.

Negated Relations: standard LaTeX	
`\neq`	$\neq$

Negated Relations: `amssymb` package							
`\nless`	$\nless$	`\nleq`	$\nleq$	`\nleqslant`	$\nleqslant$	`\nleqq`	$\nleqq$
`\lneq`	$\lneq$	`\lneqq`	$\lneqq$	`\lvertneqq`	$\lvertneqq$	`\lnsim`	$\lnsim$
`\lnapprox`	$\lnapprox$	`\ngtr`	$\ngtr$	`\ngeq`	$\ngeq$	`\ngeqslant`	$\ngeqslant$
`\ngeqq`	$\ngeqq$	`\gneq`	$\gneq$	`\gneqq`	$\gneqq$	`\gvertneqq`	$\gvertneqq$
`\gnsim`	$\gnsim$	`\gnapprox`	$\gnapprox$	`\nprec`	$\nprec$	`\npreceq`	$\npreceq$
`\precneqq`	$\precneqq$	`\precnsim`	$\precnsim$	`\precnapprox`	$\precnapprox$	`\nsim`	$\nsim$
`\nshortmid`	$\nshortmid$	`\nmid`	$\nmid$	`\nvdash`	$\nvdash$	`\nVdash`	$\nVdash$
`\ntriangleleft`	$\ntriangleleft$	`\nsubseteq`	$\nsubseteq$	`\subsetneq`	$\subsetneq$	`\varsubsetneq`	$\varsubsetneq$
`\subsetneqq`	$\subsetneqq$	`\varsubsetneqq`	$\varsubsetneqq$	`\nsucc`	$\nsucc$	`\nsucceq`	$\nsucceq$
`\succneqq`	$\succneqq$	`\succnsim`	$\succnsim$	`\succnapprox`	$\succnapprox$	`\ncong`	$\ncong$
`\nshortparallel`	$\nshortparallel$	`\nparallel`	$\nparallel$	`\nvDash`	$\nvDash$	`\nVDash`	$\nVDash$
`\ntriangleright`	$\ntriangleright$	`\ntrianglerighteq`	$\ntrianglerighteq$	`\nsupseteq`	$\nsupseteq$	`\nsupseteqq`	$\nsupseteqq$
`\supsetneq`	$\supsetneq$	`\varsupsetneq`	$\varsupsetneq$	`\supsetneqq`	$\supsetneqq$	`\varsupsetneqq`	$\varsupsetneqq$

There are many different arrows available in LaTeX and in particular in $\mathcal{AMS}$LaTeX. Arrows have the same spacing rules as binary operators.

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

Arrows: `latexsym` or `amssymb` packages	
`\leadsto`	$\leadsto$

Arrows: `amssymb` package					
`\leftleftarrows`	$\leftleftarrows$	`\leftrightarrows`	$\leftrightarrows$	`\Lleftarrow`	$\Lleftarrow$
`\twoheadleftarrow`	$\twoheadleftarrow$	`\leftarrowtail`	$\leftarrowtail$	`\looparrowleft`	$\looparrowleft$
`\leftrightharpoons`	$\leftrightharpoons$	`\curvearrowleft`	$\curvearrowleft$	`\circlearrowleft`	$\circlearrowleft$
`\Lsh`	$\Lsh$	`\upuparrows`	$\upuparrows$	`\upharpoonleft`	$\upharpoonleft$
`\downharpoonleft`	$\downharpoonleft$	`\multimap`	$\multimap$	`\leftrightsquigarrow`	$\leftrightsquigarrow$
`\rightrightarrows`	$\rightrightarrows$	`\rightleftarrows`	$\rightleftarrows$	`\Rrightarrow`	$\Rrightarrow$
`\twoheadrightarrow`	$\twoheadrightarrow$	`\rightarrowtail`	$\rightarrowtail$	`\looparrowright`	$\looparrowright$
`\rightleftharpoons`	$\rightleftharpoons$	`\curvearrowright`	$\curvearrowright$	`\circlearrowright`	$\circlearrowright$
`\Rsh`	$\Rsh$	`\downdownarrows`	$\downdownarrows$	`\upharpoonright`	$\upharpoonright$
`\restriction`	$\restriction$	`\downharpoonright`	$\downharpoonright$	`\rightsquigarrow`	$\rightsquigarrow$
`\dashrightarrow`	$\dashrightarrow$	`\dashleftarrow`	$\dashleftarrow$		

Arrows can, of course, be negated using `\not`, so `\not\rightarrow` gives $\not\rightarrow$. However, preferable results are given by a few negated arrow symbols, shown below. These are all available only in the `amssymb` package.

Negated Arrows: `amssymb` package					
`\nleftarrow`	$\nleftarrow$	`\nLeftarrow`	$\nLeftarrow$	`\nleftrightarrow`	$\nleftrightarrow$
`\nrightarrow`	$\nrightarrow$	`\nRightarrow`	$\nRightarrow$	`\nLeftrightarrow`	$\nLeftrightarrow$

In TeXnicCenter many of these symbols are available from a toolbar. In the fourth row at the top of the TeXnicCenter window, to the right of the alignment buttons, you will find the *mathbar*: a row of buttons each of which opens a toolbar containing a selection of symbols or other LaTeX features. By clicking on one of these symbols you can insert its LaTeX name into your .tex document.

19. Dots

There are a number of variations on the theme of '. . . ' in AMSLaTeX; mostly, for a horizontal series of three dots you type `\dots` and let the system work out where to put them, dependent on context. For example, in `$a_1+\dots+a_n$` which gives $a_1 + \dots + a_n$, the dots are centred vertically, whereas in `$a_1,\dots,a_n$`, which gives $a_1, \dots, a_n$, they are aligned on the baseline. If you want to force a particular style for some reason, use `\dotsb` for dots with binary operators or relations, normally centred vertically, `\dotsc` for dots with commas, normally on the baseline, `\dotsm` for multiplication dots, and `\dotsi` for dots with integrals. You should only do this if the dots fall at the end of an expression, when the system will be unable to find the appropriate style for itself, for example $a_1 + a_2 + \dotsb$ which is given by `$a_1+a_2+\dotsb$`.

Note that the exact positioning of the dots is controlled by the document class.

In ordinary LaTeX classes, no such sophistication is available. The decision is yours: `\ldots` gives . . ., and `\cdots` gives $\cdots$. You can also use these commands in AMSLaTeX styles, but the use of the commands above is preferable.

There are also the `\vdots` and `\ddots` commands, which give $\vdots$ and $\ddots$ and are mostly used in matrices (See Section 36).

In TeXnicCenter the commands `\ldots`, `\cdots`, `\vdots` and `\ddots` are available from the toolbar "White spaces and dots" (second from the left in the mathbar).

20. Superscripts and Subscripts, Primes and Operator Limits

Superscripts and subscripts are normally obtained with the ^ and _ symbols. For example, to produce $x_0y^2=z_1^3$, type `$x_0y^2=z_1^3$`. As you can see, symbols with both a superscript and a subscript are correctly handled.

If you need to script more than one symbol, enclosed the script in braces, like the argument to a macro, e.g. $2^{10}=1024$ is obtained by `$2^{10}=1024$`. Lamport [5, §2.2.1] recommends using braces to delimit all such scripts, preferring `$x^{2}+y^{2}=z^{2}$` to `$x^2+y^2=z^2$`. Knuth, however [4, Chapter 4], prefers the latter form. Both produce the same result, $x^2+y^2=z^2$; the difference is purely a matter of style. Following Knuth gives less cluttered and easier to read source files, but you may occasionally end up with e^2t when you mean e^{2t}, by typing `e^2t` instead of `e^{2t}`.

There are some non-obvious occasions when bracing is essential. The most common situation is when a subscript which looks like one symbol is internally processed as more than one: for a highly artificial example, compare $x_\neq$ with $x_{\neq}$, generated by `$x_\neq$` and `$x_{\neq}$` respectively.

One other time when bracing is essential is in a tower of superscripts or in the case of a multiple subscript (a well of subscripts?): to produce

$$n_{p_q} < n^{p^q}$$

type `$$n_{p_q} < n^{p^q}$$`; expressions like `x^y^z` generate 'double superscript' errors. You can nest symbols to any depth like this, but don't get carried away: three levels is almost certainly as far as you should go on the grounds of readability.

One symbol is automatically superscripted, the prime symbol which is just typed as `'` (apostrophe), so to produce $x''+x'=0$, type `$x''+x'=0$`.

Note that because `'` creates a superscripted symbol, something like `x'^2` will cause a double superscript error, so you need to use `{x'}^2`. You should also consider using `(x')^2`, on the grounds of readability.

For some symbols (called *operators*), limits may be typeset using subscripts for the lower limits and superscipts for upper limits, for example

$$\sum_{j=1}^n x^j = (x^{n+1}-1)/(x-1)$$

is produced by

```
\[
 \sum_{j=1}^n x^j = (x^{n+1}-1)/(x-1)
\]
```

These functions are given below:

Operators									
`\bigcap`	$\bigcap$	`\bigcup`	$\bigcup$	`\bigsqcup`	$\bigsqcup$	`\bigodot`	$\bigodot$	`\bigoplus`	$\bigoplus$
`\bigotimes`	$\bigotimes$	`\biguplus`	$\biguplus$	`\bigvee`	$\bigvee$	`\bigwedge`	$\bigwedge$	`\coprod`	$\coprod$
`\idotsint`	$\int\cdots\int$	`\iint`	$\iint$	`\iiint`	$\iiint$	`\iiiint`	$\iiiint$	`\inf`	$\inf$
`\int`	$\int$	`\lim`	$\lim$	`\liminf`	$\liminf$	`\limsup`	$\limsup$	`\max`	$\max$
`\min`	$\min$	`\oint`	$\oint$	`\prod`	$\prod$	`\sum`	$\sum$	`\sup`	$\sup$

You can also define your own operators: see Section 39. For multi-line subscripts such as

$$\sum_{\substack{n\in A\\ n^2<K}} a_n$$

see Section 21 below.

Limits are treated differently in display and inline styles: compare $\int_0^1 f$ with

$$\int_0^1 f.$$

To force a symbol to be processed as if it were in display style or text style, precede it with a `\displaystyle` or `\textstyle` declaration. For example, to produce the inline formula $\displaystyle\sum_{j=0}^n x_j^2$ you could type

```
$\displaystyle\sum_{j=0}^n x_j^2$
```

The `\textstyle` and `\displaystyle` declarations act rather like the long font-changing commands (Section 13) and the size-changing commands (Section 14): they apply to the current group. This most commonly means the end of the formula, which is usually what you want, but you can use braces to localise the effect. For example, compare

```
\[
 \textstyle \lim_{n\to\infty} x_n + \sum_{k=1}^\infty y_j
\]
```

$$\textstyle \lim_{n\to\infty} x_n + \sum_{k=1}^\infty y_j$$

with

```
\[
{\textstyle \lim_{n\to\infty} x_n} + \sum_{k=1}^\infty y_j
\]
```

$${\textstyle \lim_{n\to\infty} x_n} + \sum_{k=1}^\infty y_j$$

In the first formula, the `\textstyle` declaration has affected both the limit and the sum; in the second, it has only affected the limit.

21. Stacks

$\mathcal{AMS}$LaTeX allows us to stack formulae on top of each other by means of the (badly named) `\substack{}` command. `\substack{}` takes one argument, which consists of as many different formulae as you want, separated by double backslashes `\\`. An artificial example:

```
\[
\substack{1+1=2 \\ 2+2=4 \\ 3+3=6}
\]
```

$$\substack{1+1=2 \\ 2+2=4 \\ 3+3=6}$$

The reason for the name is that this command is most commonly used for multi-line subscripts, like this:

```
\[
\sum_{\substack{n\in A \\ n^2<K \\ n\neq 0}} a_n < 1
\]
```

$$\sum_{\substack{n\in A \\ n^2<K \\ n\neq 0}} a_n < 1$$

Note carefully the syntax: there is an underscore as well as a `\substack{}` command, and the `\substack{}` has a brace pair around it. Two common incorrect forms of the above are:

```
\[
\sum\substack{n\in\N \\ n^2<K \\ n\neq 0} a_n < 1
\]
\[
\sum_\substack{n\in\N \\ n^2<K \\ n\neq 0} a_n < 1
\]
```

(missing underscore; missing brace pair) neither of which work. You can also use `\substack{}` to produce multi-line superscripts (use a caret instead of an underscore; does anyone actually use such notation?) and, as we have seen, to produce multi-line formulae which are neither subscripted nor superscripted.

A similar stacking command is `\stackrel{}{}`, which is available in any document class. The first argument is printed in the same size as a sub- or superscript, directly above the second argument which is printed in the ordinary style. For example:

```
\[
x \stackrel{\text{def}}{=} y
\]
```

$$x \stackrel{\text{def}}{=} y$$

Notice the use of `\text{}` to produce def instead of def. `\mbox{}` would not have worked properly here; the size of font would have been wrong.

22. Fractions and Binomial Coefficients

Fractions are obtained with the `\frac{}{}` macro, which takes two arguments: the numerator and the denominator. To generate

$$\frac{1}{2}+\frac{1}{3}=\frac{5}{6}$$

type

```
\[
\frac{1}{2}+\frac{1}{3}=\frac{5}{6}
\]
```

Centring of the numerator and denominator is automatically handled: `\frac{x+y}{2}` produces $\frac{x+y}{2}$.

Binomial coefficients ($\mathcal{AMS}$ classes only) can be created with `\binom{}{}`:

$$\binom{5}{2}=\frac{5!}{3!2!}=10$$

was produced by

```
\[
\binom{5}{2}=\frac{5!}{3!2!}=10
\]
```

You might have noticed that fractions (and binomial coefficients) are treated differently in display and inline styles: the fonts used in the inline version are much smaller. Compare $\frac{1}{2}$ with

$$\frac{1}{2}$$

which were both produced by `\frac{1}{2}`. $\mathcal{AMS}$LaTeX provides `\tfrac{}{}` (text-style fraction) and `\dfrac{}{}` (display-style fraction) which produce the same output in any context, as well as the analogous commands `\tbinom{}{}` and `\dbinom{}{}`.

In ordinary LaTeX use `{\displaystyle\frac{}{}}` or `{\textstyle\frac{}{}}` to imitate `\dfrac{}{}` and `\tfrac{}{}` (note the extra level of braces to localise the effects of the style declarations). These commands are also available in $\mathcal{AMS}$LaTeX, as you saw in Section 20, but their use with fraction and binomial commands is common enough to warrant the abbreviations.

23. Roots, Square and Otherwise

The simplest use of the `\sqrt{}` macro is this: `\sqrt{x}`, which produces $\sqrt{x}$. The bar extends to cover any symbol, so $\sqrt{x^2+y^2+z^2}$ is produced by `\sqrt{x^2+y^2+z^2}`.

`\sqrt{}` takes an optional argument, given as usual in square brackets before the mandatory argument. To produce $\sqrt[3]{x}$, type `\sqrt[3]{x}`.

In $\mathcal{AMS}$ classes, the position of this optional argument can be fine-tuned using `\leftroot{}` and `\uproot{}` to move the index further away from the $\surd$ symbol. Compare:

```
\[
 \sqrt[\beta]{x}; \qquad \sqrt[\leftroot{-2}\uproot{2}\beta]{x}
\]
```

$$\sqrt[\beta]{x}; \qquad \sqrt[\leftroot{-2}\uproot{2}\beta]{x}$$

To quote the official $\mathcal{AMS}$LaTeX documentation, 'the units are a small amount that is a useful size for such adjustments.' To quote Grätzer [3, §4.4.3] 'Experiment with the arguments ... to find the best spacing for a particular root.'

In TeXnicCenter, fraction and root commands are available in the toolbar Mathematical... in the mathbar.

24. Delimiters

Delimiters are pairs of symbols like brackets. TeX provides a set of delimiters that can be scaled up to fit the size of formula between them, as follows:

Delimiters									
`(`	$($	`)`	$)$	`[`	$[$	`]`	$]$	`\{`	$\{$
`\}`	$\}$	`\lfloor`	$\lfloor$	`\rfloor`	$\rfloor$	`\lceil`	$\lceil$	`\rceil`	$\rceil$
`\langle`	$\langle$	`\rangle`	$\rangle$	`|`	$\vert$	`/`	$/$	`\backslash`	$\backslash$
`\\|`	$\Vert$	`\uparrow`	$\uparrow$	`\downarrow`	$\downarrow$	`\updownarrow`	$\updownarrow$	`\Uparrow`	$\Uparrow$
`\Downarrow`	$\Downarrow$	`\Updownarrow`	$\Updownarrow$						

To scale a delimiter pair to the right size, the `\left` and `\right` commands are used. Compare

```
\[
( \frac{1}{2} )
\]
```

$$(\frac{1}{2})$$

with

```
\[
\left( \frac{1}{2} \right)
\]
```

$$\left(\frac{1}{2} \right)$$

As you can see, in the first example the brackets have stayed at their natural size, but in the second they have scaled to fit around the fraction. All of the delimiters mentioned will scale to fit any formula.

The `\left` and `\right` command must always occur in matching pairs. However, the delimiters used with them need not match, so one can typeset

$$\left(-\frac{2}{3}, \frac{1}{3}\right]$$

with

```
\[
\left(-\frac{2}{3}, \frac{1}{3}\right]
\]
```

There is also a *null delimiter* denoted by `.` which acts as a delimiter but produces no text. This is not used very often in $\mathcal{AMS}$LaTeX but to give an artificial example:

```
\[
\left\Uparrow \frac{p}{q} \right.
\]
```

$$\Big\Uparrow \frac{p}{q}$$

In TeXnicCenter, a selection of delimiters is available in the toolbar Boundaries in the mathbar.

25. Math Accents, Over- and Underlining, Bracing and Arrowing

Math accents are similar to text accents (see Section 10), but have different control sequence names. Here are the standard accents:

Math Accents: standard LaTeX							
`\acute{x}`	$\acute{x}$	`\bar{x}`	$\bar{x}$	`\breve{x}`	$\breve{x}$	`\check{x}`	$\check{x}$
`\ddot{x}`	$\ddot{x}$	`\dot{x}`	$\dot{x}$	`\grave{x}`	$\grave{x}$	`\hat{x}`	$\hat{x}$
`\tilde{x}`	$\tilde{x}$	`\vec{x}`	$\vec{x}$	`\widehat{xx}`	$\widehat{xx}$	`\widetilde{xx}`	$\widetilde{xx}$

Math Accents: $\mathcal{AMS}$ classes only			
`\dddot{x}`	$\dddot{x}$	`\ddddot{x}`	$\ddddot{x}$

The two wide accents, `\widetilde{}` and `\widehat{}`, stretch to accomodate the formula they accent. However, there are limits to the stretchability and also to the aesthetic results: `$\widehat{f*g*h}$` gives $\widehat{f*g*h}$, not a very good result. For these extreme cases, consider alternative notation such as $(f*g*h)^\wedge$, given by `$(f*g*h)^\wedge$`.

In TeXnicCenter, a selection of math accents is available in the toolbar Accents in the mathbar.

There may be occasions when you want to have the accent symbol by itself in your document. For example, you might want to write 'let $\bar{}$ denote complex conjugation.' If you really want to do this, one way is to use a sequence like `$\bar{\phantom{M}}$`, which is what I typed above. M is the largest character in most fonts, so this gives quite a high position for the accent. Any other character can be used, if you want to fine-tune the positioning. Note, however, that `\bar{}` on its own does not work!

The `\bar{}` symbol often looks too small, especially on capital letters, for example

$$\bar{K}\cup\bar{M}=\bar{W}.$$

You might prefer to use `\overline{}` in this case, which would give

$$\overline{K}\cup\overline{M}=\overline{W},$$

and was produced by

```
\[
\overline{K}\cup\overline{M}=\overline{W}
\]
```

This will stretch to cover any formula at all:

```
\[
 \overline { \overline{a} + \overline{b} } = a + b
\]
```

$$\overline{\overline{a}+\overline{b}}=a+b$$

We also have the imaginatively named counterparts `\underline{}`, `\overleftarrow{}`, `\overrightarrow{}` which are all in standard LaTeX, `\overleftrightarrow{}`, `\underleftarrow{}`, `\underrightarrow{}`, and `\underleftrightarrow{}` which are available only in $\mathcal{AMS}$LaTeX. We illustrate some of them in another highly artificial example:

```
\[
\overleftarrow{xxx}; \quad \underleftrightarrow{yyy}; \quad
  \overleftarrow{\underrightarrow{zzz}}
\]
```

$$\overleftarrow{xxx}; \quad \underleftrightarrow{yyy}; \quad \underrightarrow{\overleftarrow{zzz}}$$

Finally, we have two similar commands: `\overbrace{}` and `\underbrace{}`. These also stretch to cover any formula, but they have the additional feature that superscripts and subscripts on the brace commands can be used to put labels on the braces. For example,

```
\[
 x^n = \underbrace{ xxx \dots x}_{n\text{ factors}}
\]
```

$$x^n = \underbrace{xxx\dots x}_{n\text{ factors}}$$

Returning to the fine details of accents, the letters i and j should not have their dots as well as accents: $\hat{i}$ looks very distracting. Dotless symbols $\imath$ and $\jmath$ are available, called `\imath` and `\jmath`; to produce $\bar{\imath}\hat{\jmath}$, for example, type `$\bar{\imath}\hat{\jmath}$`.

Double accents can come out very badly using these macros, for example `\hat{\hat{A}}` gives $\hat{\hat{A}}$, not a very desirable result. AMSLaTeX provides additional macros, with the same name but capitalised, which cope with this much better: `\Hat{\Hat{A}}` gives $\Hat{\Hat{A}}$, a great improvement. You need to use the capitalised forms for both accents.

Do not be tempted to use the capitalised forms for all purposes: they slow down processing quite considerably.

26. Math-mode Fonts

There are two common reasons for switching to a special math-mode font: to generate bold symbols, or to pull occasional symbols from another alphabet, such as calligraphic, blackboard bold, script or Fraktur (Gothic). All of this can be handled with the math-mode font-switching commands, which are very similar to the text-mode ones you have already seen (Section 13). However, note that to use the script font `\mathscr{}` you must add `mathrsfs` to the `\usepackage{}` declaration.

Here is a complete list of math-mode font-changing commands.

Math-Mode Font Switching Commands		
macro	meaning	example
`\mathbb{}`	AMS blackboard bold	$\mathbb{NZQRC}$
`\mathbf{}`	bold	$\mathbf{x^2 + y}\sin(\alpha)$
`\mathcal{}`	calligraphic	$\mathcal{ABCDEFG}$
`\mathfrak{}`	AMS Fraktur	$\mathfrak{x}^2 + \mathfrak{y}\sin(\alpha)$
`\mathit{}`	italic	$\mathit{x^2 + y}\sin(\alpha)$
`\mathnormal{}`	normal	$\mathnormal{x^2 + y}\sin(\alpha)$
`\mathrm{}`	Roman	$\mathrm{x^2 + y}\sin(\alpha)$
`\mathsf{}`	sans serif	$\mathsf{x^2 + y}\sin(\alpha)$
`\mathtt{}`	typewriter	$\mathtt{x^2 + y}\sin(\alpha)$
`\mathscr{}`	script	$\mathscr{ABCDEFG}$

Note that these commands change the style of letters and numbers; they do not change the style of function names, Greek letters or special symbols (because for most of these fonts, these symbols don't exist: asking for a Fraktur style Greek letter is meaningless).

In the calligraphic font `\mathcal{}`, the AMS blackboard bold font `\mathbb{}` and the script font `\mathscr{}`, only uppercase letters are available, no other symbols.

The blackboard bold and Fraktur styles are available only in AMS classes, the other styles are available in all LaTeX classes. To use Fraktur or blackboard bold in another class, add `amsfonts` to the `\usepackage{}` declaration.

Although you would not normally want to, it is possible to typeset an entire formula in bold, including Greek letters, special symbols and function names. To do this you use the `\boldmath` declaration *before* (not in) the formula, and `\unboldmath` after it. For example:

```
\boldmath
\[
x+y=z
\]
\unboldmath
```

$$\boldsymbol{x+y=z}$$

If you miss out the `\unboldmath` declaration, all the remaining formulae in your document will be typeset in bold. Alternatively, since `\boldmath` is a declaration whose scope is the current group, putting braces around the `\boldmath` declaration and the formula to which it refers has the same effect:

```
{\boldmath $a+b$}, $a+b$
```

$\boldsymbol{a+b}$, $a+b$

To highlight the difference between `\boldmath` and `\mathbf{}`, compare $\boldsymbol{e^{\alpha}\sin(x)}$ with $\mathbf{e}^{\alpha}\sin(\mathbf{x})$; the first was produced with `\boldmath` and the second, where only the $\mathbf{e}$ and $\mathbf{x}$ are in bold, was produced with `\mathbf{}`. Note also that `\mathbf{}` gives upright Roman bold, whereas `\boldmath` gives italic bold.

Finally, we mention the $\mathcal{AMS}$-specific `\boldsymbol{}` macro. This is used to embolden particular symbols inside a non-bold formula.

```
\[
\boldsymbol{\alpha}; \quad \alpha
\]
```

$$\boldsymbol{\alpha}; \quad \alpha$$

```
\[
\lambda+\boldsymbol{ab}
\]
```

$$\lambda+\boldsymbol{ab}$$

As the second formula shows, `\boldsymbol{}` is not restricted to individual symbols, but works with subformulae as well.

Part 5. **Document Structure**

27. Sectioning Units

An important concept in LaTeX which will be used often in the remainder of these notes, is that of an *environment*. This a block of text enclosed in a pair of delimiters `\begin{name}` and `\end{name}`, which is handled in a particular way. You have already seen an environment whose name is `\document`. TeXnicCenter provides short cuts to many environments, obtained by typing the name of the environment and then pressing the CONTROL and SPACE keys simultaneously. It is not necessary to type the whole name of the environment; as soon as you have typed enough to identify an environment uniquely, a yellow box will appear giving the name of the environment, and you can then accept this environment by pressing CTRL+SPACE. For example, if you type `doc` and then press CTRL+SPACE, TeXnicCenter will produce `\begin{document}`, a blank line, and `\end{document}`, with the cursor at the beginning of the blank line so that you can immediately start typing in this environment.

Documents are normally divided into *sectioning units*: chapters, sections and so on. The book, report and article classes all provide sections, subsections and subsubsections; the book and report classes additionally provide chapters as the highest-level standard sectioning unit.

This document is in `amsart` class, so does not have chapters, but it does have sections and subsections: the most recent heading (Sectioning Units) is a section heading. Here are subsection and subsubsection headings:

27.1. **An example subsection heading.**

27.1.1. *An example subsubsection heading.*

The section numbers are generated automatically, not typed. This means that you can reorder, delete and add sections without worrying about the numbering scheme.

The main sectioning commands are `\chapter{}`, `\section{}`, `\subsection{}` and `\subsubsection{}`; they all take one mandatory argument, the title of the section. For example, the subsubsection heading above was generated by

```
\subsubsection{An example subsubsection heading}
```

and the section heading above it came from

```
\section{Sectioning Units}
```

The `\part{}` command is slightly different: it is the highest level of sectioning, above even `\chapter{}`, but it does not affect the numbering of the other sections. Thus, Section 4 in Part 1 would be followed by Section 5 in Part 2.

The first paragraph in a section is not normally indented. To force LaTeX to indent it, begin the paragraph with the `\indent` macro mentioned in Section 6.

Section numbers are of the form 1.2.3, which would mean section 1, subsection 2, subsubsection 3 in an article style. In `amsbook` style, chapter numbers are in Roman numerals, so a section number would look like II.1 for the first section in chapter II. In general, the exact form of the number depends on the document class.

For a sectioning command, the starred form (same name but with a star after it, such as `\section*{}`) produces the title of the section, typeset in the usual way, but does not give it a number. This is often used for short sections that do not form part of the main sequence, such as a preface or short introductory section (abstracts and acknowledgements are handled separately: see Section 29).

You might also want to use unnumbered sections at the lowest level of sectioning in your document, to distinguish between short, logically distinct sections, without cluttering up the numbering scheme.

LaTeX can build a table of contents automatically from your section headings. To do this, put `\tableofcontents` where you want the table (usually at the beginning, of course). LaTeX needs to be run *twice or three times in a row* to properly generate a table of contents: the first time, it records the information; the second time, it prints it; and this might change the page sequence, requiring a third pass to get it right.

You must exercise a little care about the text you put into a section heading. A few LaTeX commands are known as *fragile*, and do not work in the argument of a sectioning command (these arguments are called *moving arguments*, because they appear in two places at once: the table of contents and the section heading).

If you want to know more about fragile commands and moving arguments, you can refer to [5] and [1]. For now, the following rule of thumb applies: if a command in a section heading generates a peculiar error message, put the command `\protect` immediately in front of it, for example:

```
\section{Summary of Earlier Work~\protect\cite{Smith95}}
```

The `\cite{}` command is discussed in Section 30.

28. Cross References and Citations

Question: when you want to refer to a textual unit which has an automatically-generated number associated with it, how do you know what number to use? Answer: you don't, you use a *label* instead. There are four commands associated with labels, all of which take one argument: `\label{}`, `\ref{}`, `\pageref{}` and `\eqref{}` (`\eqref{}` is defined only in the $\mathcal{AMS}$ document classes).

For example, this section begins with the lines

```
\section{Cross Referencing}
\label{sec:crossref}
```

Whenever LaTeX sees a command like `\label{sec:crossref}`, it associates with the argument of the `\label{}` command (in this case the label `sec:crossref`) two pieces of information: the most recent automatically-generated number, and the current page. In this case, the number is 28, generated by the `\section{}` command above it, and the page is 25.

To refer to this section from any other part of the document, the `\ref{}` and `\pageref{}` commands are used:

```
Section~\ref{sec:crossref} on page~\pageref{sec:crossref}
```

Section 28 on page 25

The `\eqref{}` command in $\mathcal{AMS}$ document classes is similar to `\ref{}`, but puts brackets around the number, so `\eqref{sec:crossref}` generates (28). It is intended, as its name suggests, to refer to automatically-generated equation numbers, not to section numbers. Environments which generate equation numbers will be described in Section 38.

The automatic cross-referencing available in LaTeX is one of its most useful and powerful features. Used wisely, it saves a great deal of time and effort. The principal advantage of using it is that you can rearrange, add or delete sections or equations, and all of the cross-referencing remains correct. It also makes your source code easier to write and edit: as you will see later, theorems and equations can all be labeled and referred to, so you can write

```
by Theorem~\ref{thm:Cauchy}
```

instead of

```
by Theorem~2.1
```

All references should routinely have ties (~ characters: see Section 12) before the referencing command, to prevent output like this:

. . . Using Theorem

1, we see that . . .

Thus, one should use

```
Section~\ref{sec:intro}
Equation~\eqref{eqn:Euler}
page~\pageref{thm:Schwarz}
```

in preference to

```
Section \ref{sec:intro}
Equation \eqref{eqn:Euler}
page \pageref{thm:Schwarz}
```

The convention used here, that section labels look like `sec:...`, equation labels like `eqn:...` and so on is by no means mandatory: you can use any text you like to label any textual unit you like. However, it is often worthwhile, especially in a long document, to adopt such a scheme, especially if you can never remember if you called that result in the introduction a Lemma or a Proposition or a Theorem.

When you use `\label{}` and `\ref{}`, you sometimes have to run LaTeX twice (or even three times) in a row to resolve all cross-references. The reason for this is that on the first pass, LaTeX associates page and section numbers with labels and on the second pass it inserts the numbers for each label `\ref`erred to. Occasionally this will upset the pagination, calling for a third pass. You can tell when this is necessary by messages like `LaTeX warning: labels may have changed. Rerun to get cross-references right`. If you `\ref{}` a label which was never declared with a `\label{}`, then you will get messages like `LaTeX warning: undeclared reference`.

You will see later (Section 30) that LaTeX provides similar facilities for automatic bibliography generation.

29. Topmatter

The *topmatter* of a document consists of such information as the title, the author's name and address, the date and so on. There are two basic ways to provide topmatter: by a `titlepage` environment or by the `\maketitle` comand.

The simpler of these two is the `titlepage` environment. This is placed immediately after the `\begin{document}` and typesets its contents on a separate page with no page number. You are entirely responsible for the contents of the `titlepage` environment: anything goes. For example, you could say something like this (the `center` environment centres its contents on the page: see Section 32)

```
\begin{titlepage}
  \begin{center} \Huge \bfseries
    Beer Drinking: A Topic In Applied Fluid Dynamics
  \end{center}
  \vspace{10ex}
  \begin{center}
    Submitted \today{} in partial fulfillment of the requirements of the
    MMath degree.
  \end{center}
  \vspace{10ex}
  {\itshape I wish to express my thanks to the landlord of the
```

```
    Charles~XII.}
\end{titlepage}
```

which would give a first page consisting solely (no page number, no running header) of

Beer Drinking: A Topic In Applied Fluid Dynamics

Submitted October 17, 2007 in partial fulfillment of the requirements of the MMath degree.

I wish to express my thanks to the landlord of the Charles XII.

The next page would be numbered page 1.

Alternatively, you can use `\maketitle`. You do this by grouping together a series of topmatter declarations after `\begin{document}` but before the beginning of the text, and putting the `\maketitle` command after them.

In ordinary LaTeX document classes, the topmatter consists of `\title`{}, `\author`{}, `\thanks`{} and `\date`{} commands, but AMSLaTeX classes provide many more options, all taking one argument. This is the full list.

`\title`{}: Title of the document. In AMS classes, `\title`{} can also take a optional argument in square brackets which is a short form of the title to be used as a running head. For example, in `amsart` style the declaration

```
\title[The Riemann Hypothesis]{A proof of the Riemann
  Hypothesis which I knocked up this lunchtime}
```

would result in the full title ('A proof of . . . ') on the front page, with the shortened form ('The Riemann Hypothesis') as the running head on odd-numbered pages (the author's name is used on even-numbered pages).

`\subjclass`{}: Subject Classification, as used in *Math. Rev.*

`\email`{}: email address of most recently named author. The address will be typeset in the typewriter family `like this`.

`\thanks`{}: Acknowledgements of support.

`\keywords`{}: Keywords, as used in *Math. Rev.*

`\address`{}: Address of most recently named author. You should separate different lines with `\\` which in `amsbook` class will be treated as line breaks, and in `amsart` class will be turned into commas.

`\translator`{}: Name of translator.

`\date`{}: Date of document. Use `\today` to put the current date in, but once the document has been finalised you should probably put the date in explicitly.

`\author`{}: Name of author. For multiple authors, use several `\author`{} declarations.

`\curradr`{}: Current address of most recently named author, if different from the address in the `\address`{} declaration.

`\dedicatory`{}: For dedications.

After these declarations, none of which actually produce any text in the output, you need a `\maketitle` command, which formats the information given in the topmatter declarations and prints a title page.

For example, here is the topmatter from this document:

```
\title{An Introduction to Mathematical Document Production Using \AmS\LaTeX}
\author{Simon Eveson}
\address{Department of Mathematics\\University of York\\
         Heslington\\York YO1 5DD}
\email{spe1@york.ac.uk}
\date{November 1994--July 1997}
\maketitle
```

For multiple authors, you should give an `\author{}` declaration for each one, and after the appropriate `\author{}` declaration, the `\address{}`, `\curraddr{}`, `\email{}` and `\thanks{}` declarations for that author.

After the `\maketitle` command, you can give an abstract in the `abstract` environment (that is, between lines reading `\begin{abstract}` and `\end{abstract}`).

```
\begin{abstract}
  In this paper we will show ...
\end{abstract}
```

If you want a table of contents, you can put `\tableofcontents` after the `\maketitle` or after the `abstract` environment. The table of contents is automatically generated from your section headings. Lists of figures and tables can be obtained with `\listoffigures` and `\listoftables` commands; see Section 35.

Like cross-references, LaTeX needs to be run twice or possibly three times in a row for the table of contents and the lists of figures and tables to be correct, because the first pass remembers where the sections start, the second pass constructs the table, and a third pass may be necessary if constructing the table has changed the page sequence.

30. Manual Bibliography Maintenance

Where one uses `\label{}` and `\ref{}` for internal cross references, one uses `\bibitem{}` and `\cite{}` for references to books or articles. The `\bibitem{}` commands are collected together inside a `thebibliography` environment, usually at the end of the document. This environment does two things: it typesets the bibliography and also defines the labels (*citation keys*) which you use to refer to the items in the bibliography. A simple bibliography looks something like this:

```
\begin{thebibliography}{9}
\bibitem{Knuth86} Knuth, D.~E., The \TeX{}book, Addison-Wesley, 1986.
\bibitem{Lamport94} Lamport, L., \LaTeX: A Document Preparation System,
  Addison-Wesley, 1994.
\end{thebibliography}
```

The environment argument `9` indicates that there are no more than nine items in the bibliography. This would be replaced with `99` for ten to a hundred entries, or `999` for up to a hundred to a thousand entries (the only significant thing about the argument is actually its width, but stick to using `9`, `99`, . . .).

The arguments to the `\bibitem{}` commands are *citation keys*, much like labels (Section 28), and are referred to using `\cite{}`, which operates in a similar way to `\ref{}` (Section 28). If the above bibliography was in use, the commands `\cite{Knuth86}` and `\cite{Lamport94}` would produce [1] and [2], respectively. Like labels, you can use any text you like as a citation key. Ties (`~`) should be routinely used for citations in the same way as they are used for cross-references, so `Knuth~\cite{Knuth86}` is preferable to `Knuth \cite{Knuth86}`, to avoid bad line breaks.

`\cite{}` takes an optional argument, which is typeset inside the square brackets with the reference number. For example, `\cite[Chapter~2]{Knuth86}` might produce [1, Chapter 2].

You can cite several sources with the same `\cite{}` command. For example, if we use the bibliography above, then `\cite{Knuth86,Lamport94}` gives [1,2]. Combining optional arguments with multiple citations is a bad idea.

Instead of numbers, you can arrange for any text you like to be used when you cite a bibliography entry. To do this, you need to use optional arguments to the `\bibitem{}` commands, like this:

```
\begin{thebibliography}{K86}
\bibitem[K86]{Knuth86} Knuth, D.~E., The \TeX{}book, Addison-Wesley, 1986.
\bibitem[L94]{Lamport94} Lamport, L., \LaTeX: A Document Preparation System,
  Addison-Wesley, 1994.
\end{thebibliography}
```

The argument to the environment should now be the longest of the labels given in the optional arguments to `\bibitem{}` commands. Now, `\cite{Knuth86}` will produce [K86] in the text.

If you are only going to produce one major document in LaTeX then this method of producing the bibliography is the easiest. However, if you plan to write several documents with common references, there is a program called BibTeX which can be a great time-saver. BibTeX is described in Section 44.

31. Theorems

In $\mathcal{AMS}$ classes, theorems are different from standard LaTeX, so everything in this section is $\mathcal{AMS}$LaTeX specific. See [5] for the standard LaTeX approach.

Theorems are declared in the preamble (between the `\documentclass{}` declaration and the `\begin{document}` line; see Section 5) with the `\newtheorem{}{}` command, which looks like this:

```
\newtheorem{lem}{Lemma}
\newtheorem{thm}{Theorem}
```

This defines environments called `lem` and `thm`, which print titles ‘Lemma’ and ‘Theorem’ when they are used. Theorems declared like this are automatically numbered; if you use `\newtheorem*{}{}` instead of `\newtheorem{}{}`, the resulting environments do not generate numbers.

To control the numbering of theorems, `\newtheorem{}{}` can take optional arguments. To define a theorem numbered within each section instead of throughout the document (so numbers will look like 1.1), use

```
\newtheorem{thm}{Theorem}[section]
```

To make two differently-named theorem environments follow the same numbering scheme, use

```
\newtheorem{lem}{Lemma}
\newtheorem{prop}[lem]{Proposition}
```

Now, Lemma 1 will be followed by Proposition 2.

For example, if the following command

```
\newtheorem{thm}{Theorem}[section]
```

is used, then theorems look like this:

```
\begin{thm} \label{thm:binom}
  For all $x,y\in\mathbb{C}$ and $n\in\mathbb{N}$,
  \[
  (x+y)^n=\sum_{r=0}^n\binom{n}{r}x^ry^{n-r}.
  \]
\end{thm}
```

Theorem 31.1. *For all $x, y \in \mathbb{C}$ and $n \in \mathbb{N}$,*

$$(x+y)^n=\sum_{r=0}^n\binom{n}{r}x^ry^{n-r}.$$

Theorems defined this way take an optional argument, which is printed after the title:

```
\begin{thm}[Cauchy] \label{thm:Cauchy}
  If $\gamma$ is a contour in an open set $\Omega\subseteq\mathbb{C}$
  and $f$ is analytic in $\Omega$ then
  \[
  \int_\gamma f(z)dz = 0.
  \]
\end{thm}
```

Theorem 31.2 (Cauchy). *If γ is a contour in an open set $\Omega \subseteq \mathbb{C}$ and f is analytic in Ω then*

$$\int_\gamma f(z)dz = 0.$$

The labels are to make it easy to refer to the relevant theorem as explained in Section 28.

There are three visual styles available for theorems: `plain`, which is the default, `definition` and `remark`. The exact kind of output is class-dependent, but here are theorems in the three styles in the current class (`amsart`):

Theorem 31.1. *This is a plain-style theorem.*

$$1+1=2.$$

Theorem 31.1. This is a definition-style theorem.

$$1+1=2.$$

Theorem 31.1. This is a remark-style theorem.

$$1 + 1 = 2.$$

To use the different styles, use `\theoremstyle{}` declarations as follows:

```
\newtheorem{thm}{Theorem}
\newtheorem{lem}{Lemma}

\theoremstyle{definition}
\newtheorem{defn}{Definition}

\theoremstyle{remark}
\newtheorem{rem}{Remark}
```

This defines `thm` and `lem` as plain-style (because `plain` is the default), `defn` as definition-style and `rem` as remark-style.

There is a `proof` environment already defined in $\mathcal{AMS}$LaTeX. It is headed *Proof*, its body is typeset in the same style as the running text, and a Λ symbol is put at the end, for example:

```
\begin{proof}
  Obvious.
\end{proof}
```

Proof. Obvious. □

`proof` takes an optional argument, which is used instead of (not in addition to) the *Proof* heading:

```
\begin{proof}[Proof (necessity)]
  Blindingly obvious.
\end{proof}
```

Proof (necessity). Blindingly obvious. □

Part 6. **Textual Displays and Structures**

32. Centred and Flushed Text

The `center` (note spelling) environment centres its text in the page, for example

```
\begin{center}
  Displayed Title
\end{center}
```

Displayed Title

The environments `flushleft` and `flushright` typeset their contents flush to the left margin and flush to the right margin, respectively:

```
\begin{flushleft}
  Left margin
\end{flushleft}
\begin{flushright}
  Right margin
\end{flushright}
```

Left margin

Right margin

Just as in paragraph mode, spaces and line breaks in your input file are ignored, and if the content of the environment is too long to fit on one line it is broken up and each line is centred or flushed individually. This paragraph is being tyeset inside a `flushleft` environment: notice the ragged right edge and the lack of indentation at the beginning.

If you want to provide explicit line breaks, use the `\\` command:

```
\begin{center}
  Displayed Title \\
  Date
\end{center}
```

Displayed Title
Date

You should *never* put a `\\` after the last line of a centred or flushed environment (or indeed, any environment). It goes *between* lines, not after lines.

TeXnicCenter provides buttons in the toolbar for the three alignment environments. These look the same as in most word-processors. Clicking on any of them gives `\begin{}` and `\end{}`, with the cursor on a blank line between them. If you have highlighted a section of text, it will be placed between the `\begin{}` and `\end{}` lines.

Note that an environment constitutes a TEX group, so the effect of any declaration made inside an environment is normally limited to the environment. For example,

```
\begin{center}
  \Large \bfseries A large, bold, centred phrase.
\end{center}
```

A large, bold, centred phrase.

The `\Large` and `\bfseries` declarations (Sections 14, 13) are scoped to apply only to the `center` environment, so they do not affect any text outside it and do not need to be enclosed in a brace pair as they would in running text.

33. Enumerated and Itemised Lists

An *enumerated list* looks like this:

```
\begin{enumerate}
\item Apologies for absence.
\item Minutes of the previous meeting.
\item Matters arising.
\item Any other business.
\item Down the pub.
\end{enumerate}
```

(1) Apologies for absence.
(2) Minutes of the previous meeting.
(3) Matters arising.
(4) Any other business.
(5) Down the pub.

Note that the numbers were not typed, but automatically generated. Each separate item in the list is introduced by an `\item` command, and is typeset in paragraph mode but indented away from the numbers.

You might want to use numbers in parentheses to label the items. You can arrange this for the whole document by typing

```
\renewcommand{\labelenumi}{(\theenumi)}
```

in the preamble. You can restrict this kind of labelling to a specific instance of the `enumerate` environment as follows.

```
\begin{enumerate}
\renewcommand{\labelenumi}{(\theenumi)}
\item Apologies for absence.
\item Minutes of the previous meeting.
\item Matters arising.
\item Any other business.
\item Down the pub.
\end{enumerate}
```

(1) Apologies for absence.
(2) Minutes of the previous meeting.
(3) Matters arising.

(4) Any other business.
(5) Down the pub.

You can also achieve this by putting the desired label in square brackets immediately after the `\item` command.

```
\begin{enumerate}
\item[(1)] The first thing.
\item[(2)] The second thing.
\item[(3)] And finally.
\end{enumerate}
```

(1) The first thing.
(2) The second thing.
(3) And finally.

You can have enumerated lists within enumerated lists, in which case the second level is numbered using letters (in this document class, anyway). You can have up to four levels of enumeration in total.

For lists within lists, you type something like this:

```
\begin{enumerate}
\item This is just an ordinary item.
\item
  \begin{enumerate}
  \item This is the first item in the sublist.
  \item and so it goes on \dots
  \end{enumerate}
\end{enumerate}
```

(1) This is just an ordinary item.
(2) (a) This is the first item in the sublist.
    (b) and so it goes on . . .

The `itemize` (note spelling) environment is used in exactly the same way as the `enumerate` environment, but instead of being numbered, the items are labelled.

```
\begin{itemize}
\item The labels are normally bullets,
\item like this,
\item
  \begin{itemize}
  \item but if you use sublists then other symbols
  \item are employed.
  \end{itemize}
\end{itemize}
```

• The labels are normally bullets,
• like this,
• – but if you use sublists then other symbols
  – are employed.

You can also have enumerated lists within itemised lists, and vice-versa.

The final list-making environment is the `description` environment, in which the labels are not automatically generated but supplied by the user. A `description` environment produces output like this:

```
\begin{description}
\item[Enumerated list] Items are labelled 1,2,3,\dots
\item[Itemized list] Items are labelled with bullets.
\item[Description] User supplies labels.
\end{description}
```

Enumerated list: Items are labelled 1,2,3,. . .
Itemized list: Items are labelled with bullets.
Description: User supplies labels.

The label is placed between square brackets immediately after the `\item` command. This is an example of an *optional argument* in LaTeX, similar to the mandatory arguments you have already seen but enclosed in square brackets instead of braces. As the name suggests, optional arguments may be omitted but in this case the output will look rather strange if they are.

These list environments can all be typed quickly in TeXnicCenter, as described at the beginning of Part 5.

34. Tables

Tables are produced by the `tabular` environment in paragraph mode and by the `array` environment in math mode. Both environments take a mandatory argument after `\begin{tabular}` or `\begin{array}`. This argument consists of one letter for each column of the table, either `l` if the entries are to be left-aligned in the column, `c` if they are to be centred or `r` if they are to be right-aligned. Additionally, the character `|` can be used at any point to indicate a vertical line between columns, or `||` for two vertical lines, and so on.

Inside either environment, columns are separated by `&` characters and rows by `\\` (like in the centre and flush environments described in Section 32). A horizontal line between two rows is obtained by putting `\hline` after the `\\` between the rows. To put a horizontal line before the first row, put the `\hline` immediately after the argument to `\begin{tabular}`, and to put a line after the last row, put `\\ \hline` after it. Note that this is the only time you ever put a `\\` after the last row of a table! You can put several horizontal lines, separated by a small amount of vertical space, by repeating the `\hline` commands. Vertical lines generated by `|` in the environment argument do not extend through this space.

`tabular` environments are normally used inside another environment, such as `center` (Section 32) or `table` (Section 35). If you don not put them inside an environment like this to control where they will appear on the page, strange positioning is very likely. On the other hand, `array` environments are normally used in display-style mode.

Some examples:

```
\begin{center}
  \begin{tabular}{||r|c|l||} \hline
    This  & example   & has     \\ \hline \hline
    three & columns   & aligned \\ \hline \hline
    in    & different & ways    \\ \hline \hline
    with  & lines     & between \\ \hline \hline
    them  & and       & double  \\ \hline \hline
    lines & between   & rows.   \\ \hline
  \end{tabular}
\end{center}
```

This	example	has
three	columns	aligned
in	different	ways
with	lines	between
them	and	double
lines	between	rows.

Although you do not have to format the rows and columns in your source file to reflect the final shape of the table, it makes the source much easier to read if you do. Of course, this may not be possible, especially if the table is very large or complicated, but it's worth trying.

Tables generated with the commands above have exactly the same column structure in each row. A common requirement is to have one or more rows treated differently, for example to hold the title of the table. The `\multicolumn{}{}{}` command allows a single piece of text to spread out over several columns. It takes three arguments: the number of columns, the alignment, and the text. For example,

```
\begin{center}
  \begin{tabular}{|r|r||r|r|} \hline
    \multicolumn{4}{|c|}{Table Of Squares} \\ \hline
    1 &  1 & 2 &  4 \\
    3 &  9 & 4 & 16 \\
    5 & 25 & 6 & 36 \\ \hline
  \end{tabular}
\end{center}
```

Table Of Squares			
1	1	2	4
3	9	4	16
5	25	6	36

In this example, the `\multicolumn{}{}{}` command extends across four columns, is centred, and contains the words ‘Table of Squares’. Note the use of `|` characters in the `\multicolumn{}{}{}` command.

The`array` environment might be used to produce the multiplication table of a group or a table of integrals. For example,

```
\[
\begin{array}{|c|cccc|}\hline
       & 1      & \rho   & \sigma  & \tau  \\ \hline
  1    & 1      & \rho   & \sigma  & \tau  \\
\rho   & \rho   &   1    &  \tau   & \sigma \\
\sigma & \sigma & \tau   &    1    & \rho   \\
\tau   & \tau   & \sigma &  \rho   &   1 \\ \hline
\end{array}
\]
```

	1	ρ	σ	τ
1	1	ρ	σ	τ
ρ	ρ	1	τ	σ
σ	σ	τ	1	ρ
τ	τ	σ	ρ	1

```
\[
\begin{array}{|c|c|}\hline
 F'(x)=f(x)               &    F(x)=\int f(x)dx  \\ \hline
x^a\quad \quad (a\neq 1) &    \dfrac{x^{a+1}}{a+1} + C\\
x^{-1}                   &    \ln x + C \\
e^x                      &    e^x + C  \\
\sin x                   &    -\cos x  + C \\ \hline
\end{array}
\]
```

$F'(x)=f(x)$	$F(x)=\int f(x)dx$
$x^a\quad \quad (a\neq 1)$	$\dfrac{x^{a+1}}{a+1} + C$
x^{-1}	$\ln x + C$
e^x	$e^x + C$
$\sin x$	$-\cos x + C$

Tables can be inserted in TeXnicCenter in the usual way for an environment, or by clicking on the Tabular icon on the toolbar, which allows you to choose the alignment and position options.

Tables can be much more complicated than this. See [5, §§3.6.2,C.10.2] for many other options such as horizontal lines spanning particular columns and vertical lines extending only the height of one row. For much more sophisticated table environments, see [1].

35. Floats: Tables and Figures

Floats are objects which are not necessarily typeset at the current position in the document, but are allowed to move about (float) to fit into a convenient position. This is normally because they are so large that there is no guarantee that they will fit onto the current page if they are typeset here and now. (Look back at the symbol tables earlier in these notes to see what happens if you don't use the float mechanism and insist on inserting big tables in fixed places!) Because you do not know where in the document they will finally appear, they are usually labelled with a number and a caption, and referred to with `\ref{}` or `\pageref{}` (see Section 28).

The two environments described here are `table` and `figure`. Neither of these produce any text by themselves; they just cause their contents to float.

A `table` environment normally contains a `tabular` environment (see Section 34), and looks something like this:

```
\begin{table}
  \begin{tabular}{...}
    ...
  \end{tabular}
  \caption{Demonstration Table}
  \label{tab:demo}
\end{table}
```

The `\caption{}` commands typesets an automatically-generated number for the table, followed by its argument; in this case, 'Demonstration Table.' Since the `\caption{}` command generates the number, not the `table` environment, the `\label{}` command must always come *after* the `\caption{}` command.

`figure` environments are similar, but are intended to encompass some kind of graphic image. Various ways of producing some types of graphics are considered in Sections 40, 41 and 42. Often, however, the simplest way of producing a relevant picture or diagram (from TEX's point of view, anyway) is just to leave some space and photocopy or draw the image in afterwards. For this, you need a `\vspace{}` command (see Section 11) to reserve some vertical space. A `figure` environment would look something like this:

```
\begin{figure}
  \vspace{50mm}
  \caption{Demonstration Figure}
  \label{fig:demo}
\end{figure}
```

This leaves 50mm of vertical space in which to put the picture. Again, the `\label{}` command comes after the `\caption{}` command.

If you want a list of figures or a list of tables, you can put a `\listoffigures` or `\listoftables` command at any point in your document, usually at the beginning after the topmatter (see Section 29) and the `\tableofcontents` command.

The argument to a `\caption{}` command is technically a *moving argument* (see Section 27), so some commands (*fragile commands*) used in it may have to be protected. Without going too far into the technicalities of this, if an apparently innocuous LATEX command in a caption heading causes obscure error messages, try putting `\protect` immediately in front of it, e.g.

```
\caption{Illustration From~\protect\cite{BookInLibrary}}
```

Part 7. **Mathematical Displays and Structures**

36. Matrices

Matrices are one of the areas of major difference between AMSLATEX and ordinary LATEX. Everything in this section is AMSLATEX specific: for LATEX's (less convenient and less visually attractive, but more flexible) versions of these commands, refer to [5, §3.3.3].

There are six matrix environments: `matrix`, `pmatrix`, `bmatrix`, `vmatrix`, `Vmatrix` and `smallmatrix`. They all have exactly the same syntax, which should be quite familiar by now: rows are separated (not terminated) by `\\`, and entries in each row are separated by `&`. Unlike the `tabular` environment, there is no argument to specify column alignments: all columns are centred (if you need more exotic alignments, see the `array` environment in [5, §3.3.3]). For example,

```
\[
\begin{bmatrix}
  a & b \\
  c & d
\end{bmatrix}
\]
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

You need to be in math mode when you use any of these environments, and for all except the `smallmatrix` environment you should probably be in display math mode.

The forms differ only in the delimiters around them: `matrix` just typesets the matrix, with no brackets around it, `pmatrix` uses round brackets (p for parentheses), `bmatrix` uses square brackets, `vmatrix` uses single vertical lines, and `Vmatrix` uses double vertical lines. Thus, repeating the above example with the first five variants gives:

$$\begin{matrix} a & b \\ c & d \end{matrix} \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \begin{vmatrix} a & b \\ c & d \end{vmatrix} \qquad \begin{Vmatrix} a & b \\ c & d \end{Vmatrix}$$

(The space in between the matrices was created with the spacing command `\qquad`, which gives a space the same width as MM in the current font. See Section 11.)

If you want other delimiters, use a combination of `\left`, `\right` and `matrix`:

```
\[
  \left\{
    \begin{matrix}
      a & b \\
      c & d
    \end{matrix}
  \right\}
\]
```

$$\left\{ \begin{matrix} a & b \\ c & d \end{matrix} \right\}$$

The `smallmatrix` environment, which does not produce any delimiters, is intended for small matrices to go into the running text, for example $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ was produced by

```
$\left(\begin{smallmatrix}
  1 & 0 \\
  0 & 1
\end{smallmatrix}\right)$
```

You are allowed up to ten columns in these matrix environments. If you need more, you can say, for example,

```
\setcounter{MaxMatrixCols}{15}
```

to increase the allowed number to 15.

37. THE `cases` ENVIRONMENT

Structures like

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

are produced by the `cases` environment. The above example came from

```
\[
f(x) =  \begin{cases}
             1 & x>0, \\
             0 & x=0, \\
            -1 & x<0.
          \end{cases}
\]
```

As usual, rows are separated by `\\`, and columns by `&`. The `\text{}` macro is often used to insert text on the right-hand side:

```
\[
g(t) =  \begin{cases}
            \dfrac{\sin(t)}{t} & \text{if } t\neq 0 \\
            1                  & \text{if } t=0.
          \end{cases}
\]
```

$$g(t) = \begin{cases} \dfrac{\sin(t)}{t} & \text{if } t\neq 0 \\ 1 & \text{if } t=0. \end{cases}$$

Two points to note: the space after ‘if’ in `\text{if }` is interpreted literally by the `\text{}` macro, and the formulae on the right are typeset in text style, not display style, so `\dfrac{}{}` was used to force the fraction into its display style.

38. Displayed and Aligned Formulae

LaTeX has a number of ways of aligning and tagging equations; AMSLaTeX has many more. In this section, everything except the `equation` environment is AMSLaTeX specific. Ordinary LaTeX has counterparts to some of these features, but by no means all: see [5, §3.3].

The simplest display environment is `equation`. This is similar to using `\[` and `\]` to enter and exit from display math mode, but `equation` automatically supplies a number (usually called a *tag*) for the equation, for example:

```
\begin{equation}
  \sum_{n=1}^\infty\frac{1}{n^2} = \frac{\pi^2}{6}
\end{equation}
```

$$\sum_{n=1}^\infty\frac{1}{n^2} = \frac{\pi^2}{6} \tag{1}$$

Tags are placed on the left margin by default; to put them on the right margin like this, use the `reqno` documentclass option. The next equation will receive the next number in sequence:

```
\begin{equation}
  \sum_{n=1}^\infty\frac{1}{n^4} = \frac{\pi^4}{90}
\end{equation}
```

$$\sum_{n=1}^\infty\frac{1}{n^4} = \frac{\pi^4}{90} \tag{2}$$

and so on. To refer to these numbers later, use `\label{}` in exactly the same way as for section numbers:

```
\begin{equation}
\label{eqn:euler}
  e^{i\pi}=-1
\end{equation}
```

$$e^{i\pi}=-1 \tag{3}$$

Now, the `\eqref{}` command can be used to refer to the equation: `\eqref{eqn:euler}` gives (3). The commands `\ref{}` and `\pageref{}` also work in the usual way (see Section 28), giving the number without brackets and the page on which the reference occurs.

Equations can be given non-numeric tags with the `\tag{}` and `\tag*{}` commands, for example:

```
\begin{equation}
  \int_\infty^\infty e^{-x^2}dx = \pi \tag{$\dag$}
\end{equation}
\begin{equation}
  \int_1^\infty\frac{dt}{t^2} = 1 \tag*{$\ddag$}
\end{equation}
```

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

$$\int_1^\infty\frac{dt}{t^2} = 1 \tag*{$\ddag$}$$

The difference between `\tag`{} and `\tag*`{} is just that `\tag`{} puts brackets around the tag, and `\tag*`{} does not.

Note that the argument of a `\tag` command is interpreted as text, not maths, so to use the math-mode symbols † and ‡ above, there is a `$` pair inside the argument of the `\tag` command.

To go with the `equation` environment there is an `equation*` environment, which does not produce an automatic number. This is essentially equivalent to using `\[` and `\]`.

For an equation which is too long to fit on one line, the `multline` environment can be used (note spelling: `multline`, not `multiline`). Breaks in the equation are indicated using `\\`. For example,

```
\begin{multline}
  \left( \sum_{n=0}^\infty a_nz^n \right) \cdot
  \left( \sum_{n=0}^\infty b_nz^n \right) +
  \left( \sum_{n=0}^\infty c_nz^n \right) \cdot
  \left( \sum_{n=0}^\infty d_nz^n \right) + \\
  \left( \sum_{n=0}^\infty e_nz^n \right) \cdot
  \left( \sum_{n=0}^\infty f_nz^n \right) \\
  = \sum_{n=0}^\infty \sum_{k=0}^n (a_kb_{n-k}+c_kd_{n-k}+e_kf_{n-k}) z^n
\end{multline}
```

$$\begin{multline}\left(\sum_{n=0}^\infty a_nz^n \right) \cdot \left(\sum_{n=0}^\infty b_nz^n \right) + \left(\sum_{n=0}^\infty c_nz^n \right) \cdot \left(\sum_{n=0}^\infty d_nz^n \right) + \\ \left(\sum_{n=0}^\infty e_nz^n \right) \cdot \left(\sum_{n=0}^\infty f_nz^n \right) \\ = \sum_{n=0}^\infty \sum_{k=0}^n (a_kb_{n-k}+c_kd_{n-k}+e_kf_{n-k}) z^n \tag{4}\end{multline}$$

As you can see, the first line is aligned on the left, the last line is aligned on the right, and the other lines are centred.

There is a `multline*` environment to go with `multline`, which does not automatically tag the equation. In either form, `\tag`{} or `\tag*`{} can be used to provide an individual tag for the equation, like in an `equation` environment.

When several equations follow in sequence, it is convenient to use a `gather` or `gather*` environment rather than multiple `equation` environments. The equations are separated by `\\`, as usual. In a `gather*` environment, the equations are not numbered unless you do so with explicit `\tag`{} or `\tag*`{} command; in a `gather` environment, they are all numbered individually. To suppress numbering on an individual element of a `gather` environment, use `\notag`. The commands `\tag`{} and `\notag` are both illustrated below:

```
\begin{gather}
  1 + 1 = 2 \\
  2 + 2 = 4 \notag \\
  4 + 4 = 8 \tag{$*$} \\
  8 + 8 = 16
\end{gather}
```

$$1+1=2 \tag{5}$$
$$2+2=4$$
$$4+4=8 \tag{$*$}$$
$$8+8=16 \tag{6}$$

The `align` environment is used for a sequence of equations which should be aligned with each other, typically to keep $=$ or $<$ signs under each other. The equations are separated by `\\` as usual, and each one should contain one & character to indicate the point of alignment. When the alignment is set, the points indicated by & are all in line vertically. For example,

```
\begin{align}
(A+B)^2 &= (A+B)(A+B) \\
        &= A(A+B)+B(A+B) \\
        &= A^2+AB+BA+B^2
\end{align}
```

$$\begin{aligned}(A+B)^2 &= (A+B)(A+B) && (7)\\ &= A(A+B)+B(A+B) && (8)\\ &= A^2+AB+BA+B^2 && (9)\end{aligned}$$

As in the `gather` environment, there is an `align*` environment which does not automatically tag the equations, tags on individual equations can be suppressed using `\notag` and equations can be given particular tags with `\tag`{} or `\tag*`{}.

Within an alignment, lines of text can be inserted without disturbing the alignment structure, using the `\intertext`{} command. This is placed immediately after a `\\`, *and does not have another* `\\` *after it*. For example,

```
\begin{align*}
  \|x+y\|^2 &=    \|x\|^2+\|y\|^2+2\operatorname{Re}\langle x,y \rangle \\
            &\leq \|x\|^2+\|y\|^2+2\|x\|\|y\| \\
\intertext{(by the Cauchy-Schwarz inequality)}
            &=    (\|x\|+\|y\|)^2
\end{align*}
```

$$\begin{aligned}\|x+y\|^2 &= \|x\|^2+\|y\|^2+2\operatorname{Re}\langle x,y\rangle\\ &\leq \|x\|^2+\|y\|^2+2\|x\|\|y\|\end{aligned}$$

(by the Cauchy-Schwarz inequality)

$$= (\|x\|+\|y\|)^2$$

The `align` environment can also typeset several columns, each with its own alignment point. The columns are separated by & and the lines as usual separated by `\\`. For example, to give three parallel aligned columns we proceed as follows:

```
\begin{align}
  x_1 &= 1 & y_1 &= 2 & z_1 &= 3 \\
  x_2 &= 2 & y_2 &= 3 & z_2 &= 4
\end{align}
```

$$\begin{aligned}x_1 &= 1 & y_1 &= 2 & z_1 &= 3 && (10)\\ x_2 &= 2 & y_2 &= 3 & z_2 &= 4 && (11)\end{aligned}$$

In an `align` structure like this, there is equal space between columns and between the display and the margins. The environment `flalign` (flush align) has identical syntax to `align` but gives equal space between columns and no space at the margins. Here is the above example generated by `flalign` instead of `align`:

$$x_1 = 1 \qquad y_1 = 2 \qquad z_1 = 3 \quad (12)$$
$$x_2 = 2 \qquad y_2 = 3 \qquad z_2 = 4 \quad (13)$$

Both environments have `*` forms to suppress automatic tagging, and the tagging on individual equations can be controlled using `\notag` and `\tag{}` as usual.

If an equation within an alignment structure or a `gather` is too long to fit on one line, you can use the `split` environment, which is similar to `multline`. Note, though, that `split` can be used only inside another environment, usually `gather` or one of the `align` structures. As in the `multline` environment, the equation is split at a `\\`.

There are other more complicated alignment structures available in $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$LaTeX. For details see [3, Chapter 5].

To control the numbering of equations, use the `\numberwithin{}{}` command in the preamble of your document. Equations normally run 1,2,3,... throughout the document, but if you wanted them numbered within sections you could put

```
\numberwithin{equation}{section}
```

in the preamble. `section` could, of course, be replaced by `chapter` or `subsection` or any other sectioning unit.

39. Defining Your Own Macros

You can define your own macros to save typing, using `\newcommand{}{}`. Macros defined this way have exactly the same status as those which are defined in LaTeX itself. For example, if you get fed up with typing `\varepsilon` to get ε, putting

```
\newcommand{\eps}{\varepsilon}
```

in the preamble means you can use `\eps` instead. Macros can be as complicated as you like: if you keep on using the phrase ‘functional analysis’ (don’t we all?) you could

```
\newcommand{\fa}{functional analysis}
```

and then type `\fa` instead. Beware: typing something like

```
\fa is exceptionally interesting.
```

doesn’t work: you need `\fa{} is` or `\fa\ is` to give a space after the macro has been expanded (see Section 9). Beware also of using too many abbreviations like this: your document source can easily become unreadable.

If you need a new log-like function (see Section 17) or operator (Section 20), then `\newcommand{}{}` can define one for you, in conjunction with `\operatorname{}` or `\operatorname*{}`. For example,

```
\newcommand{\arcsinh}{\operatorname{arcsinh}}
```

defines an `\arcsinh` macro that behaves exactly like the existing `\arcsin` one. If you want to define an operator which takes limits in the same way as `\sup`, `\lim`, etc, then use `\operatorname*{}`:

```
\newcommand{\esup}{\operatorname*{ess.sup}}
```

Now,

```
\[
\|f\|_\infty = \esup_{s\in S} |f(s)|
\]
```

$$\|f\|_\infty = \operatorname*{ess.sup}_{s\in S} |f(s)|$$

Note that `\operatorname{}` is $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$LaTeX specific; to do this in ordinary LaTeX, look up `\mathop` in [4]. $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$LaTeX also supplies a slightly more convenient shorthand for defining operators: the two examples above can also be coded as

```
\DeclareMathOperator{\arcsinh}{arcsinh}
\DeclareMathOperator*{\esup}{ess.sup}
```

One common requirement is to define short names for symbols pulled out of another math alphabet, such as $\mathcal{U}$ or $\mathbb{Z}$. One way is to do the obvious:

```
\newcommand{\Z}{\mathbb{Z}}
\newcommand{\U}{\mathcal{U}}
```

and now `$\Z$` gives $\mathbb{Z}$ and `$\U$` gives $\mathcal{U}$. This is slightly inconvenient, because the symbol only works in math mode which makes common phrases like 'the neighbourhood $\mathcal{U}$' a little awkward to type. This problem can be overcome by using the `\ensuremath{}` macro:

```
\newcommand{\C}{\ensuremath{\mathbb{C}}}
```

Now `\C` gives $\mathbb{C}$ in both text and math modes but remember to type `\C\` or `\C{}` in text mode if you want a space after $\mathbb{C}$.

If you try to define a macro with a name (e.g. `\a`) which is already used for something else, LaTeX will complain "Command `\a` already defined". If you think you will never need the existing definition, you can override it by using `\renewcommand` instead of `\newcommand` (but it might be a good idea to find out what your proposed name already means before you redefine it).

40. Commutative Diagrams

There are various ways of drawing commutative diagrams in LaTeX, and opinions differ as to which is the best. Here we describe some of the simpler features of Paul Taylor's `diagrams` package. Some other approaches are considered in Section 42. For a description of yet more packages, see [1].

To go into much detail of what this extensive and powerful system can do is beyond the scope of these notes, so we shall content ourselves with some illustrative examples of the most common shapes. For more details on what can be done, you should consult the official documentation, which is available online.

To load the package, put the command

```
\usepackage[PostScript=dvips]{diagrams}
```

in your preamble. The option `[PostScript=dvips]`, which tells the package what system we will be using to print our output, is not mandatory: you can just use `\usepackage{diagrams}`. However, putting it in slightly improves the output in some cases.

Rectangular diagrams are the simplest to draw. The diagram is generated by a `diagram` environment, which looks very much like a `matrix` environment. Every cell in the diagram can contain one of three things: a formula, an arrow, or nothing at all. Arrows are produced by four commands: `\rTo`, `\lTo`, `\uTo` and `\dTo`, meaning arrows pointing left, right, up and down. For example, here is the simplest possible diagram:

```
\begin{diagram}
  A    & \rTo & B    \\
  \dTo &      & \dTo \\
  C    & \rTo & D
\end{diagram}
```

$$\begin{tikzcd} A \arrow[r] \arrow[d] & B \arrow[d] \\ C \arrow[r] & D \end{tikzcd}$$

This is not a very realistic example: one would almost inevitably want to label the arrows. This is achieved by subscripts and superscripts on the arrow-generating commands. On a horizontal arrow (`\rTo` or `\lTo`) these do the obvious things: a superscript appears above the arrow and a subscript appears below it. On a vertical arrow (`\uTo` or `\dTo`) the convention is that a superscript goes on the left and a subscript on the right. Adding some labels to the example above gives

```
\begin{diagram}
  A      & \rTo^f & B      \\
  \dTo^h &        & \dTo_k \\
  C      & \rTo_g & D
\end{diagram}
```

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[d, "k"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

We can also add labels to the right or left of vertical (or diagonal) arrows by using > or < respectively. In the example we would have

```
\begin{diagram}
  A      & \rTo^f & B      \\
  \dTo<h &        & \dTo>k \\
  C      & \rTo_g & D
\end{diagram}
```

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[d, "k"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

You can have as many rows and columns as you need. Moreover, the arrows are not of fixed length: they extend through adjacent blank cells until they meet a cell with a formula in it. For example

```
\begin{diagram}
  A    & \rTo &   B  & \rTo & C    \\
  \dTo &      &      &      & \dTo \\
  D    &      & \rTo &      & D
\end{diagram}
```

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & B \arrow[r] & C \arrow[d] \\
D \arrow[rr] & & D
\end{tikzcd}$$

Notice how the arrow at the bottom has extended.

Non-rectangular diagrams are specified using much the same syntax, with the four additional arrows `\ruTo`, `\rdTo`, `\luTo`, `\ldTo` to give arrows pointing respectively right and up, right and down, left and up, and left and down. This allows us to generate triangles quite easily:

```
\begin{diagram}[nohug]
  A        & \rTo^{f}  & B \\
  \dTo^{h} & \ruTo_{g} &   \\
  C        &       &
\end{diagram}
```

$$\begin{tikzcd}A \arrow[r, "f"] \arrow[d, "h"'] & B \\ C \arrow[ur, "g"'] & \end{tikzcd}$$

The optional argument `[nohug]` is to ensure that the label g is not rotated. If we do not use it but do use `[PostScript=dvips]`, we would get the following.

```
\begin{diagram}
  A        & \rTo^{f}  & B \\
  \dTo^{h} & \ruTo_{g} &   \\
  C        &       &
\end{diagram}
```

$$\begin{tikzcd}A \arrow[r, "f"] \arrow[d, "h"'] & B \\ C \arrow[ur, "g"' sloped] & \end{tikzcd}$$

The advantage of using `[PostScript=dvips]` is that, without it, diagonal lines do not stretch in the same way as horizontal lines. The `diagrams` package is well capable of stretching diagonals, and of drawing lines at arbitrary angles to connect arbitrary cells, but the complexity of syntax necessitated by this flexibility is beyond the scope of these notes. Refer to Paul Taylor's documentation if you need to know more.

41. Drawing Automata and Graphs

Our description of the package `autograph` covers only the rudimentary ideas. To find out more, consult

`http://www.liafa.jussieu.fr/~gastin/autograph0.4/autograph0.4.html`

and click on doc autograph.ps(380Ko).

The package is used in conjunction with the package `epic` and to load both, put the command

```
\usepackage{epic,autograph}
```

in the preamble.

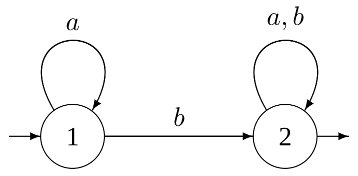
A state transition graph of an automaton is drawn within a `picture` environment which will normally be inside another environment such as `center`. After the command `\begin{picture}` you determine the size of a box to put your picture in and a coordinate system based on that box. Thus you might write `\begin{picture}(80,55)(-20,-15)`. This will give you a box 80 units across and 55 units high; the coordinates of the bottom left hand corner and the top right hand corner are (-20,-15) and (60,40) respectively. The unit of length is 4pt, that is, about 1.4mm. The second pair of coordinates may be omitted and this is equivalent to writing (0,0).

An example of a very simple automaton is the following.

```
\begin{center}
\begin{picture}(36,20)
\letstate A=(8,5) \drawinitialstate(A){1}
\letstate B=(28,5) \drawfinalstate(B){2}

\drawloop[t](A){$a$}    \drawloop[t](B){$a,b$}

\drawtrans(A,B){$b$}
\end{picture}
\end{center}
```

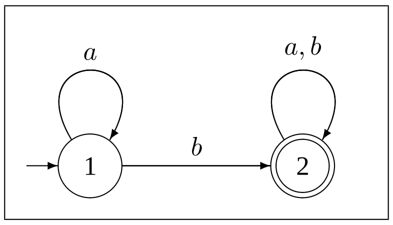


Note that states are created by the `\letstate` command. The state is given a name which does not appear in the picture and its centre is specified. To make the state appear and give it a label, use the `\drawstate` command. In the example, we used `\drawinitialstate` and `\drawfinalstate`. Not surprisingly, the former gives an initial state (indicated by the arrow entering the state from the left) and the latter gives a final state (indicated by the arrow leaving the state on the right). Final states are sometimes indicated by double circles and we do this in the following example which also shows how to enclose the picture in a box.

```
\begin{center}
\begin{picture}(36,20)
\put(0,0){\framebox(36,20){}}
\letstate A=(8,5) \drawinitialstate(A){1}
\letstate B=(28,5) \drawrepeatedstate(B){2}

\drawloop[t](A){$a$}    \drawloop[t](B){$a,b$}

\drawtrans(A,B){$b$}
\end{picture}
\end{center}
```



The `\drawloop` command takes three arguments. The first is one of the letters `t,b,l,r` and specifies whether the loop will be on the top, bottom, left or right of the state. This argument is optional and the default is `t` so that we need not have included it in the example. The second argument specifies the state and the third is the label on the loop.

The command `\drawtrans(A,B){$b$}` gives us the arrow from from the state named A to the state named B with label b. Notice that the label is on the top of the arrow, that is, as we look along the arrow the label is to the left. To have it on the right as we look along the arrow we would have used `\drawtrans[r](A,B){$b$}`.

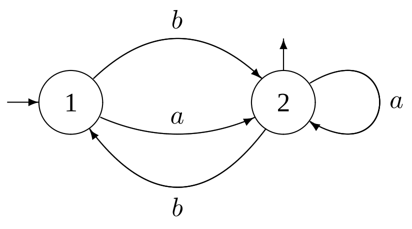
Sometimes we want to join states by curved arrows. To do this we use `\drawcurvedtrans`. We might also want to use `\setprofcurve` to determine how curved the arrow is as in the following examples.

```
\begin{center}
\begin{picture}(40,22)
\letstate A=(8,12) \drawinitialstate(A){1}
\letstate B=(28,12) \drawfinalstate[t](B){2}

\drawloop[r](B){$a$}

\setprofcurve{6} \drawcurvedtrans(A,B){$b$}
\setprofcurve{-3} \drawcurvedtrans(A,B){$a$}
\setprofcurve{8} \drawcurvedtrans(B,A){$b$}

\end{picture}
\end{center}
```



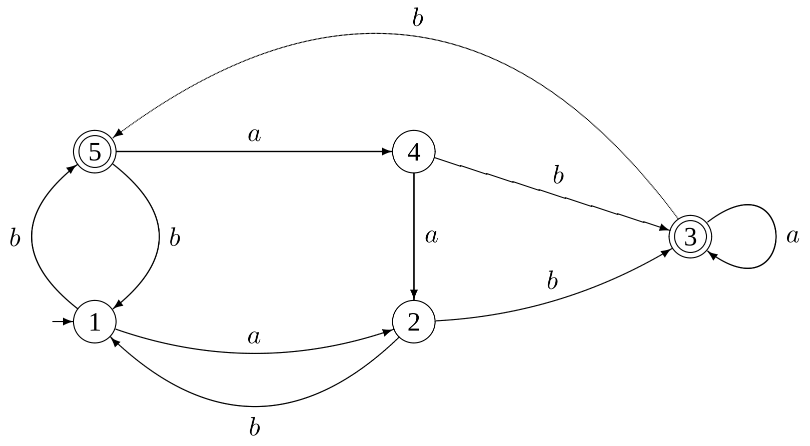
The integer in the argument of `\setprofcurve` is the distance in units of length of the middle of the transition arrow from the straight line joining (the centres of) the states. A positive (negative) value means that the transition arrow is to the left (right) with respect to the direction of transition.

We can vary the size of the circles representing states by using the command `\setstatediam{}` where the argument is an integer from 1 to 10. If the argument is n, then the circle has diameter $4n$pt, that is, about $1.4n$mm. The default value of n is 6. There is a similar command `\setrepeatedstatediam{}` which fixes the diameter of the inner of the two circles. The default value of the argument is 5.

```
\begin{center}
\begin{picture}(80,34)
\setstatediam{4}
\setrepeatedstatediam{3}

\letstate A=(8,10) \drawinitialstate(A){1}
\letstate B=(38,10) \drawstate(B){2}
\letstate C=(64,18) \drawrepeatedstate(C){3}
\letstate D=(38,26) \drawstate(D){4}
\letstate E=(8,26) \drawrepeatedstate(E){5}
\drawloop[r](C){$a$}

\setprofcurve{6} \drawcurvedtrans(A,E){$b$}
\setprofcurve{-3} \drawcurvedtrans(A,B){$a$}
\setprofcurve{8} \drawcurvedtrans(B,A){$b$}
\setprofcurve{-2} \drawcurvedtrans(B,C){$b$}
\setprofcurve{-15} \drawcurvedtrans[r](C,E){$b$}
\drawtrans(D,B){$a$}
\drawtrans(D,C){$b$}
\drawtrans(E,D){$a$}
\setprofcurve{6} \drawcurvedtrans(E,A){$b$}
\end{picture}
\end{center}
```



42. PSTRICKS

PSTricks is a general purpose package which can do much more than either `diagrams` or `autograph`. The penalty is that it is more complicated to use. We will illustrate it with some simple applications; a much fuller account can be found in Chapter 4 of [2].

To use the package you must load it by putting the command

```
\usepackage{pstricks}
```

in the preamble. In fact, there are a number of subsidiary packages and you may want to load some of these at the same time. For our examples it will be enough to put the command

```
\usepackage{pstricks,pst-node,pst-plot}
```

in the preamble. As mentioned in Section 5, `pstricks` will not load if it is listed after `a4` or `a4wide`. It is, therefore, best to put the above command before any other `\usepackage{}` commands.

You cannot use the output profile LaTeX=>PDF to process a .tex file containing graphics produced by pstricks, and TeXnicCenter will complain that it can't render the document properly if you use LaTeX=>DVI, though it will do its best. For best results, use LaTeX=>PS; if you need a .pdf output file, use LaTeX=>PS=>PDF.

42.1. **Coordinates.**

The `pspicture` environment entails choosing a coordinate system as illustrated below.

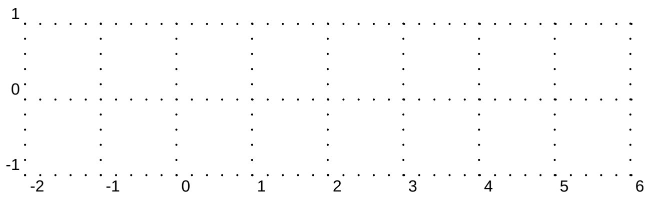
```
\begin{pspicture}(-1,0)(6,5)
\end{pspicture}
```

The unit length is 1cm and the above gives you an area 7cm wide by 5cm high in which to draw your picture. Note that in contrast to the `autograph` package, the bottom left hand corner has coordinates (-1,0) and the top right hand corner has coordinates (6,5). The picture will be drawn flushleft unless you use the environment inside another such as `center`. You can produce a dotted line grid of 1cm coordinate squares by using the command `\showgrid` which is defined as follows.

```
\newpsobject{showgrid}{psgrid}{subgriddiv=1,griddots=5,gridlabels=6pt}
```

This command would usually go in the preamble. We then get the grid as follows.

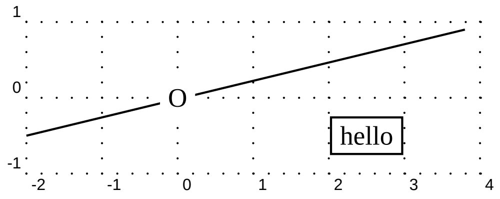
```
\begin{center}
\begin{pspicture}(-2,-1)(6,1) \showgrid
\end{pspicture}
\end{center}
```



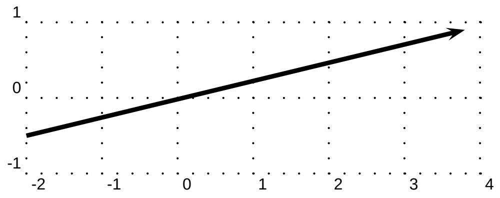
Of course, you may not want the grid in the final version of your picture but it can be helpful while you are experimenting with the layout. You can place text or other things at a specific point by using the command `\rput`. Use `\rput*` if you want to block out anything underneath whatever you are putting in place. You can rotate the material as illustrated in some of the following examples. There is a variant `\uput` which puts material a distance 5pt from the specified point; you must specify the direction by giving an angle in degrees. See Example 4 below. You can also put a frame around the material with the command `\psframebox`. Line segments can be drawn by using the command `\psline` and specifying the coordinates of the end points. Various other commands for graphics objects are shown in the examples.

Some simple examples

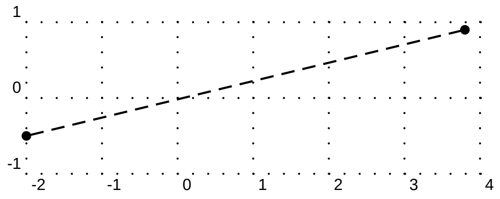
(1)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,1) \showgrid
\psline(-2,-.5)(3.8,.9)
\rput*(0,0){O}
\rput(2.5,-.5){\psframebox{hello}}
\end{pspicture}
\end{center}
```



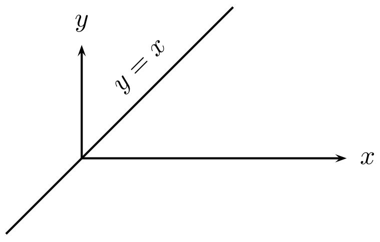
(2)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,1) \showgrid
\psline[linewidth=0.6mm]{->}(-2,-.5)(3.8,.9)
\end{pspicture}
\end{center}
```



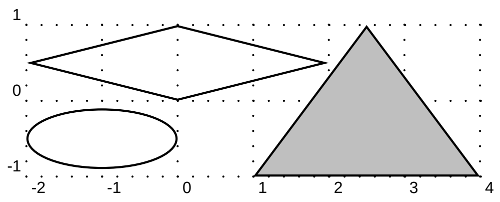
(3)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,1) \showgrid
\psline[linestyle=dashed]{*-*}(-2,-.5)(3.8,.9)
\end{pspicture}
\end{center}
```



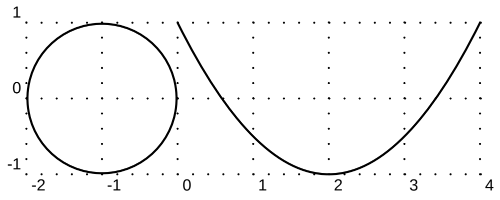
(4)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,2) %\showgrid
\psline{<->}(0,1.5)(0,0)(3.5,0)
\psline(-1,-1)(2,2)
\uput[135]{45}(1,1){$y=x$}
\uput[0](3.5,0){$x$}      \uput[90](0,1.5){$y$}
\end{pspicture}
\end{center}
```



(5)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,1) \showgrid
\psdiamond(0,.5)(2,.5)
\pstriangle[fillstyle=solid,fillcolor=lightgray](2.5,-1)(3,2)
\psellipse[fillstyle=solid](-1,-.5)(1,.4)
\end{pspicture}
\end{center}
```

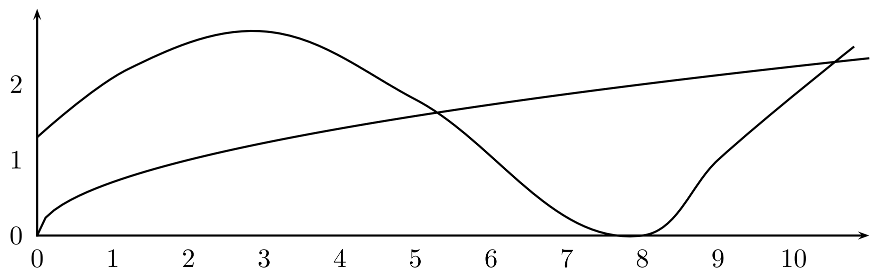


(6)
```
\begin{center}
\begin{pspicture}(-2,-1)(4,1) \showgrid
\pscircle(-1,0){1}
\parabola(4,1)(2,-1)
\end{pspicture}
\end{center}
```



We can also draw curves by specifying some coordinates that we want the curve to go through. The command `\psplot` allows the graphs of some functions to be drawn (if you know tha PostScript code for the function!) and we can produce coordinate axes with the command `\psaxes`.

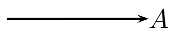
```
\begin{center}
\begin{pspicture}(0,0)(11,3) %\showgrid
\pscurve(0,1.3)(1.2,2.2)(3,2.7)(5,1.8)(8,0)(9,1)(10.8,2.5)
\psplot[plotpoints=100]{0}{11}{x 2 div .5 exp}
\psaxes[ticks=none]{->}(11,3)
\end{pspicture}
\end{center}
```



42.2. **Nodes and connections.** We consider two of the three ways of creating named nodes and joining them with labelled connections.

42.2.1. *Using coordinates.* This approach is similar in some respects to `autograph` but the commands are different and you can do much more. As with autograph each node is given a name which does not appear in the picture and the position of the node is given by a coordinate pair. There are several different commands for nodes. The simplest is `\pnode` which simply produces a point which can be joined to another node. To have text at a node you can use the commands `\rput` and `\rnode`. For example,

```
\begin{pspicture}(0,0)(3,1)
\pnode(.5,.5){A}
\rput(2.5,.5){\rnode{B}{$A$}}
\ncline{->}{A}{B}
\end{pspicture}
```



This example also shows the simplest way of connecting two nodes using `\ncline`. The arrowhead is produced by `{->}` which is optional. A wide range of arrow styles is available. We can draw commutative diagrams using these commands but to stop the arrows actually joining the nodes we use the parameter `nodesep=`*dim*.

```
\begin{pspicture}(0,0)(3,3)
\psset{nodesep=3pt}
\rput(.5,2.5){\rnode{A}{$A$}}
\rput(2.5,2.5){\rnode{B}{$B$}}
\rput(.5,.5){\rnode{C}{$C$}}
\rput(2.5,.5){\rnode{D}{$D$}}
\ncline{->}{A}{B}
\ncline{->}{B}{D}
\ncline{->}{A}{C}
\ncline{->}{C}{D}
\end{pspicture}
```

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & B \arrow[d] \\
C \arrow[r] & D
\end{tikzcd}$$

To put some labels on the arrows, the simplest commands to use are `\Aput` and `\Bput`.

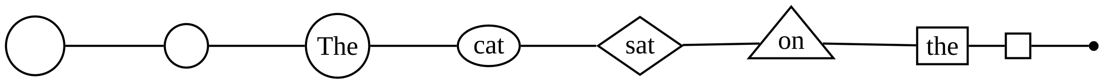
```
\begin{pspicture}(0,0)(3,3)
\psset{nodesep=3pt}
\rput(.5,2.5){\rnode{A}{$A$}}
\rput(2.5,2.5){\rnode{B}{$B$}}
\rput(.5,.5){\rnode{C}{$C$}}
\rput(2.5,.5){\rnode{D}{$D$}}
\ncline{->}{A}{B}\Aput{$f$}
\ncline{->}{B}{D}\Aput{$k$}
\ncline{->}{A}{C}\Bput{$h$}
\ncline{->}{C}{D}\Bput{$g$}
\end{pspicture}
```

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[d, "k"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

You can have circles, ovals, diamonds, triangles, dots or squares at the nodes. The relevant commands are `\cnode`, `\Cnode`, and `\circlenode` for circles, and `\ovalnode`, `\dianode`, `\trinode`, `dotnode` and `\fnode`. The last one gives a square. In each case you have to give coordinates for the centre and give the node a name but note that you use `\rput` to fix the positions of nodes created by any of `\circlenode`, `\ovalnode`, `\dianode` or `\trinode`. You cannot have text at the node when you use `\cnode`, `\Cnode`, `dotnode` or `fnode` but text is allowed with all the others. We can also get text in a rectangle by using `rnode` and `psframebox`. `\cnode` takes the radius as a parameter but with `\Cnode` the radius is specified beforehand. In the following example, the command `\psset{}` is used to fix the radius of nodes created by `\Cnode`.

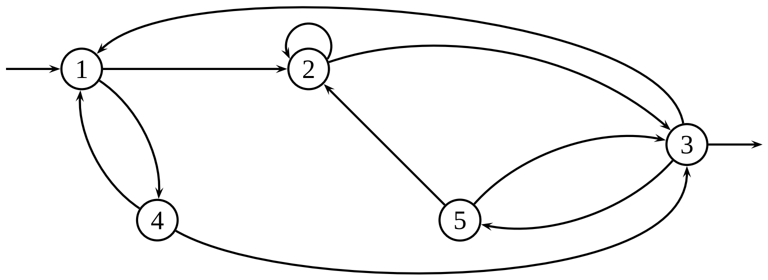
```
\begin{pspicture}(-7.5,-1)(7.5,1) %\showgrid
\psset{radius=.3}
\cnode(-7,0){.4}{A}
\Cnode(-5,0){B}
```

```
\rput(-3,0){\circlenode{C}{The}}
\rput(-1,0){\ovalnode{D}{cat}}
\rput(1,0){\dianode{E}{sat}}
\rput(3,.18){\trinode{F}{on}}
\rput(5,0){\rnode{G}{\psframebox{the}}}
\fnode(6,0){H}
\dotnode(7,0){I}
\ncline{A}{B}    \ncline{B}{C}    \ncline{C}{D}    \ncline{D}{E}
\ncline{E}{F}    \ncline{F}{G}    \ncline{G}{H}    \ncline{H}{I}
\end{pspicture}
```



As well as different styles of node there are several different ways of joining nodes. We illustrate three of them (\nccurve, \ncarcand \nccircle) in the following example.

```
\begin{pspicture}(-7,-2)(7,2) %\showgrid
\pnode(-4,1){A}
\rput(-3,1){\circlenode{B}{1}}
\rput(0,1){\circlenode{C}{2}}
\rput(5,0){\circlenode{D}{3}}
\pnode(6,0){E}
\rput(-2,-1){\circlenode{F}{4}}
\rput(2,-1){\circlenode{G}{5}}
\ncline{->}{A}{B}   \ncline{->}{D}{E}   \ncline{->}{B}{C}    \ncline{->}{G}{C}
\ncarc[arcangle=30]{->}{B}{F}    \ncarc[arcangle=30]{->}{F}{B}
\ncarc[arcangle=30]{->}{G}{D}    \ncarc[arcangle=30]{->}{D}{G}
\ncarc[arcangle=30]{->}{C}{D}    \nccurve[ncurv=.5,angleA=-30,angleB=270]{->}{F}{D}
\nccircle{->}{C}{.3} \nccurve[angleA=100,angleB=45,ncurv=.4]{->}{D}{B}
\end{pspicture}
```



We could put labels on the arrows by using the commands `\Aput` and `\Bput`. Note that `\ncarc` and `\nccurve` take optional parameters. `arcangle` is the angle (in degrees) the arc makes with the straight line joining the nodes. The default is 8. `angleA` (`angleB`) is the angle the curve makes at the first node (second node). The default in both cases is 0. `ncurv` determines how curved the curve is; the larger the number, the more curved the curve. The default is 0.67.

42.2.2. *The matrix approach.* Here we use a `psmatrix` environment which is similar to an ordinary `matrix` environment. The node connections are drawn as in the coordinate approach except that the nodes are identified by their matrix position rather than a name. Notice the method of putting labels on the connections.

```
\begin{center}
\begin{pspicture}(0,0)(4,3)
\begin{psmatrix}[nodesep=3pt]
```

```
  A    &   B  \\
  C    &   D
\ncline{->}{1,1}{1,2}^{$f$}
\ncline{->}{1,2}{2,2}>{$k$}
\ncline{->}{1,1}{2,1}<{$h$}
\ncline{->}{2,1}{2,2}_{$g$}
\end{psmatrix}
\end{pspicture}
\end{center}
```

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[d, "k"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

Note that we used the option `nodesep` to give some space between the nodes and the connectors. Labels can also be added by using the commands `\naput` and `\nbput`. For example

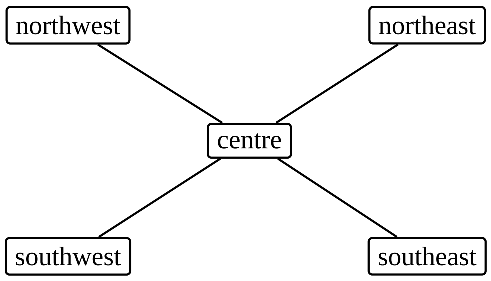
```
\begin{center}
\begin{pspicture}(0,0)(4,2.7)
\begin{psmatrix}[nodesep=3pt]
  A    &   B  \\
  C    &   D
\ncline{->}{1,1}{1,2}\naput{$f$}
\ncline{->}{1,2}{2,2}\naput{$k$}
\ncline{->}{1,1}{2,1}\nbput{$h$}
\ncline{->}{2,1}{2,2}\nbput{$g$}
\end{psmatrix}
\end{pspicture}
\end{center}
```

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[d, "k"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

As a final example we have the following.

```
\begin{center}
\begin{pspicture}(0,0)(6,3.5)
\psset{framearc=.2}
\begin{psmatrix}[rowsep=1cm,colsep=1cm]
\psframebox{northwest}  &                    & \psframebox{northeast} \\
                          & \psframebox{centre} &  \\
\psframebox{southwest}  &                    & \psframebox{southeast}
\ncline{1,1}{2,2}   \ncline{1,3}{2,2}
\ncline{3,1}{2,2}   \ncline{3,3}{2,2}
\end{psmatrix}
\end{pspicture}
```

```
\end{center}
```



In the example the command `\psset{framearc=.2}` produces rounded corners on the boxes. The way in which the parameters `rowsep` and `colsep` are used is clear. The default value is 1.5cm in both cases.

Part 8. **Other Files and Programs**

43. USING MULTIPLE FILES

For a large document, such as a thesis, it is convenient to split the document into several files, one for each chapter. The best way of organising this is to have a *master file* which has no text in it at all. In place of the text, there are `\include{}` commands, which load in the files for each chapter. A master file might look something like this:

```
\documentclass[12pt]{amsart}

\begin{document}

\include{intro}
\include{survey}
\include{results}
\include{conj}

\end{document}
```

This is very slim for a master file: things like topmatter, macro and theorem definitions, `\tablefcontents` declarations and so on would normally be here too.

The effect of this is that when LaTeX runs, the `\include{}` commands are replaced by the entire contents of the file to which they refer, in this case `intro.tex` followed by `survey.tex`, `results.tex` and `conj.tex`.

You can speed up your document production work by working on each chapter individually, and only running one file at a time through LaTeX, by using the `\includeonly{}` command in the preamble, which tells LaTeX to ignore some `\include{}` commands, like this:

```
\documentclass[12pt]{amsart}

\includeonly{intro}

\begin{document}

\include{intro}
\include{survey}
\include{results}
\include{conj}

\end{document}
```

Running this through LaTeX results in only the file `intro.tex` being processed. However, any cross-references defined in the other files are still used.

When `intro.tex` is finished, changing the `\includeonly{}` declaration to `\includeonly{survey}` allows you to work on the file `survey.tex` independently of the others, and so on. When the whole document is finished, deleting or commenting out the `\includeonly{}` declaration allows the whole document to be processed at once.

Note that, because of the way page references are stored, a file read in by an `\include` command always starts on a new page. If you do not want this, then replace `\include` by `\input`.

44. BibTeX

BibTeX is a powerful bibliography maintenance system, which automatically generates `thebibliography` environments (see Section 30). The general idea is that you create a bibliography database, which contains all the citations you ever make, and when you prepare a particular document the BibTeX system extracts only the necessary information from the database. If you are only producing one major LaTeX document, it is probably easier to use the manual bibliography commands described in Section 30. If, however, you are going to write several documents all referring to similar bibliographies then BibTeX is a major time-saver.

A bibliography file has the extension `.bib` and is normally located either in the same directory as the document, if it is only to be used with one document, or more commonly in your personal TeX macros directory (`~/tex/inputs` under Unix, `m:\tex\inputs` under MS-DOS/MS-Windows). In this location, it can always be found by BibTeX, whatever document you are working on.

A BibTeX source file consists of a sequence of *records*, each describing a book or paper which can be cited. The record for the TeXbook looks like this:

```
@book{Knuth86,
  author =         "Knuth, D. E.",
  title =          "The \TeX{}book",
  publisher =      "Addison-Wesley",
  year =           "1986"
}
```

This record begins with `@book`, indicating that it describes a book (there are other possibilities, including `@article`, `@proceedings`, `@collection` and quite a few more described below). The text after `@book` is the *citation key*, used with `\cite{}` as described above.

Within the record, there are a number of *fields*, in this case `author`, `title`, `publisher` and `year`. These are more or less self-explanatory, but note the following points:

- Fields are of the form `name = "value"`, with the value enclosed in a pair of double quotes.
- Fields are separated by commas. If you miss out a comma or put a comma after the last field, difficult-to-interpret errors may result.
- The collection of fields is enclosed in a brace pair.

To use your database, you put `\cite{}` commands into the text as usual, but instead of putting a `thebibliography` environment at the end of the document, you use the `\bibliographystyle{}` and `\bibliography{}` commands, like this:

```
\bibliographystyle{style}
\bibliography{file}
```

where `file` is the name of your bibliography file (or several files, separated with commas), and `style` is one of the available bibliography styles, chosen from the following list:

`plain`: LaTeX plain style. Citations in the text look like [1], entries in the bibliography are all in Roman, with volume 1, number 2, pages 3–4 being typeset like 1(2):3–4. Bibliography sorted according to author.

`alpha`: LaTeX alpha style. The bibliography is the same as `plain`, but the citations in the test look like [Knu86] for the TeXbook (Knuth 1986).

`amsplain`: $\mathcal{AMS}$ version of LaTeX plain style. Citations in the text are the same as `plain`, but the bibliography formatting is different (the same as the $\mathcal{AMS}$ journals use, still sorted by author).

`amsalpha`: $\mathcal{AMS}$ alpha style. Citations as in `alpha`, bibliography entries as in `amsplain`

`unsrt`: LaTeX unsorted. The same as `plain`, but with the bibliography sorted by the order of citation, not by author.

Whenever you add or delete a citation, you have to follow the following sequence of operations: first, run LaTeX on your file, then run BibTeX, then run LaTeX again *twice*. The bibliography is now accurate, and will remain so until you add or remove another citation. Don't bother keeping your system in phase until you need to print a reasonably accurate draft; if you're not proofreading on the level of checking exact references, you may as well let it stay out of phase.

By default, BIBTEX is run automatically in TeXnicCenter when you click on "Build current file". This is the reason for three of the apparent error messages mentioned on page 4. It is harmless, but if you want to stop it click on Build, then Define Output Profiles . . . , and tick the box Do not use BIBTEXin this profile.

Here is a complete list of record types. See below for information about all the different fields possible.

Most of the fields are reasonably self-explanatory, and there are notes on the use of some of the more obscure fields. The fields are described in more detail below.

`@article`: An article in a journal. Mandatory fields: `author`, `title`, `journal`, `year`. Optional fields: `volume`, `number`, `pages`, `month`, `note`.

`@book`: A book with a named publisher. Mandatory fields: `author` or `editor`, `title`, `publisher`, `year`. Optional fields: `volume` or `number`, `series`, `address`, `edition`, `note`.

`@booklet`: A bound work without a named publisher or organisation. Mandatory fields: `title`. Optional fields: `author`, `howpublished`, `address`, `month`, `year`, `note`.

`@inbook`: Part of a book, such as a chapter or a range of pages (see also `@incollection` below and `@book` above). Mandatory fields: `author` or `editor`, `title`, `chapter` and/or `pages`, `publisher`, `year`. Optional fields: `volume` or `number`, `series`, `type`, `address`, `edition`, `month`, `note`. The `type` field is used for referring to, say, a Section instead of a Chapter: use `chapter = 3` and `type = Section` to refer to Section 3.

`@incollection`: An individual work, with its own title, published as part of a collection. Mandatory fields: `author`, `title`, `booktitle`, `publisher`, `year`. Optional fields: `volume` or `number`, `series`, `type`, `chapter`, `pages`, `address`, `edition`, `month`, `note`. The `type` field serves the same purpose as in `@inbook` above. `title` is for the title of the individual work being cited, `booktitle` is for the title of the collection as a whole.

`@inproceedings`: An article in the proceedings of a conference (see also `@proceedings` below). Mandatory fields: `author`, `title`, `booktitle`, `year`. Optional fields: `editor`, `volume` or `number`, `series`, `pages`, `address`, `month`, `organization`, `publisher`, `note`.

`@manual`: A technical manual. Mandatory fields: `title`. Optional fields: `author`, `organization`, `address`, `edition`, `month`, `year`, `note`.

`@mastersthesis`: Master's thesis. Mandatory fields: `author`, `title`, `school`, `year`. Optional fields: `type`, `address`, `month`, `note`. The `type` field is used to specify the type of degree, e.g. M. Sc. or M. Phil.

`@misc`: A last resort, if none of the other types seem appropriate. Mandatory fields: none. Optional fields: `author`, `title`, `howpublished`, `month`, `year`, `note`.

`@phdthesis`: Doctoral thesis. Mandatory fields: `author`, `title`, `school`, `year`. Optional fields: `type`, `address`, `month`, `note`. The `type` field is used to specify the type of degree, e.g. Ph. D. or D. Phil.

`@proceedings`: The proceedings of a conference (see also `@inproceedings` above). Mandatory fields: `title`, `year`. Optional fields: `editor`, `volume` or `number`, `series`, `address`, `month`, `organization`, `publisher`, `note`.

`@techreport`: Technical report, such as a numbered preprint. Mandatory fields: `author`, `title`, `institution`, `year`. Optional fields: `type`, `number`, `address`, `month`, `note`.

`@unpublished`: An unpublished work with an identifiable author and title. Mandatory fields: `author`, `title`, `note`. Optional fields: `month`, `year`. Use the `note` field to guide the reader on how to obtain a copy.

Here is a complete list of fields and their meanings:

`address`: Address of publisher of a book or institution issuing a preprint or awarding a degree.

`author`: Name of author or authors. Multiple authors are separated by `and`; initials or forenames should be either after the surname, separated from it by a comma, or before the surname. For example,

```
author = "Knuth, D. E. and L. Lamport"
```

BIBTEX takes care of ties in the output, so there is no need to put them into the record.

In principle, almost any LATEX commands can go into the `author` field, but in practice this may confuse BIBTEX. It is wise to put a brace pair around any macros which you use, like this:

```
author = "G{\"{o}}del, K."
```

`booktitle`: This is used only when citing a part of a book with its own title, such as an article in a collection or conference proceedings, when `booktitle` is used for the title of the book and `title` (see below) for the title of the article. See `title` below for some notes on how to use this field.

`chapter`: The number of a chapter or section within a book.
`crossref`: See [5, B.1.4].
`edition`: Edition of the book, in the form `First`, `Second` etc. (note capitalisation; BIBTEX might switch to lower case, according to the current style).
`editor`: The name of the editor or editors, in the same format as `author` above.
`howpublished`: in the `@misc` and `@booklet` types, how the item was published.
`institution`: Name of an institution issuing a preprint or a report (for theses, use `school`, see below).
`journal`: Name of a journal.
`key`: See [5, B.2].
`language`: ($\mathcal{AMS}$ styles only). The published language of the work, as an indication that the title and author have been translated or transliterated.
`month`: Month of publication; these should be abbreviated to jan, feb, mar, etc., and BIBTEX will treat them according to the current bibliography style.
`note`: Any further information you wish to provide.
`number`: Number of an issue of a journal, or of a report or preprint. For journals, `number` is subordinate to `volume` (see below); if a journal issue has only one number associated with it, call it a `volume`.
`organization`: Organisation sponsoring a conference or publishing a manual.
`pages`: Page or range of pages. Use `--` to indicate a range, e.g. `2--6`. Several distinct pages should be separated with commas, e.g. `2,6--11,14`.
`publisher`: Name of publisher.
`school`: For theses, the University awarding the degree.
`series`: Name of a series of books.
`title`: Title of work. You should capitalise most words in a title, except odd prepositions and conjunctions; the bibliography style may decide to make some words appear in lower case. To protect letters from being converted into lower case, enclose them in a brace pair, for example `{B}anach space` or `{T}oeplitz operator`.
`type`: Used for various purposes in `@techreport`, `@inbook`, `@incollection` and `@thesis` types.
`volume`: Volume of journal or book. See also `volume` above.
`year`: Year of publication.

45. INDEXING AND MAKEINDEX

It is possible, but difficult, to maintain an index manually (see [5] if you really want to). There is a program called `makeindex` which does much of the job automatically.

The first stage in using `makeindex` is to prepare your document in the following way: firstly, if you are **not** using one of the $\mathcal{AMS}$ classes, add `makeidx` to your `\usepackage{}` declaration; secondly, put the command `\makeindex` in the preamble (i.e. before the `\begin{document}` command); finally, put the command `\printindex` at the point in your document where you want the index to appear; typically, just before the `\end{document}` command. Note that if you use an $\mathcal{AMS}$ class (`amsart` or `amsbook`) and also put `makeidx` in your `\usepackage{}` declaration, this will cause errors.

Now, whenever you want an item to appear in the index you put a `\index{}` command in your file, like this:

```
... a function is called \emph{linear}\index{linear} if ...
```

This produces an entry under ‘linear’ in the index, pointing to the page where the `\index{}` command was interpreted. Note that an `\index{}` command generates no text in the document, only in the index. The index entry for the above would look like

linear 55

To create subitems, use `!`; for example:

```
... a function is called \emph{linear}\index{function!linear} if ...
```

which produces an index entry like this :

function
    linear 55

To create the index, follow these steps: run LATEX on your file, run `makeindex` then rerun LATEX. You now have an index.

By default, `makeindex` is run automatically in TeXnicCenter when you click on “Build current file”. This is the reason for the fourth apparent error message mentioned on page 4. It is harmless, but if you want to stop it click on Build, then Define Output Profiles . . . , and tick the box Do not use MakeIndex in this profile.

There are many more things you can do with `makeindex`; see [5, Appendix A] and [1, Chapter 12] for more details.

References

[1] Frank Mittelbach and Michel Goossens, *The LaTeX companion* (2nd ed.) Addison-Wesley, 2004.
[2] Michel Goosens, Sebastian Rahtz, and Frank Mittelbach, *The LaTeX graphics companion*, Addison-Wesley, 1997.
[3] George Grätzer, *Math into LaTeX* (3rd ed.), Birkhäuser Springer, 2000.
[4] D. E. Knuth, *The TeXbook*, Addison-Wesley, 1986.
[5] L. Lamport, *LaTeX: a document preparation system, user guide and reference manual*, 2nd ed., Addison-Wesley, 1994.
[6] ______, *LaTeX: a document preparation system, user guide and reference manual*, 1st ed., Addison-Wesley, 1986.

INDEX

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF YORK, HESLINGTON, YORK YO1 5DD
E-mail address: spe1@york.ac.uk