A Gentle Introduction to TEX

A Manual for Self-study

Michael Doob
Department of Mathematics
The University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2

MDOOB@UOFMCC.BITNET
MDOOB@CCU.UMANITOBA.CA

Introduction

First the bad news: TEX is a large and complicated program that goes to extraordinary lengths to produce attractive typeset material. This very complication can cause unexpected things to happen at times. Now the good news: straightforward text is very easy to typeset using TEX. So it's possible to start with easier text and work up to more complicated situations.

The purpose of this manual is to start from the very beginning and to move towards these more complicated situations. No previous knowledge of TEX is assumed. By proceeding a section at a time, greater varieties of text can be produced.

Here are a few suggestions: there are some exercises in each section. Be sure and do them! The only way to learn TEX is by using it. Better yet, experiment on your own; try to make some variations on the exercises. There is no way that you can damage the TEX program with your experiments. You can find a complete answer to most exercises by looking at the TEX source file that was used to produce this document. You'll notice that there are references in the right margin to **The TEXbook**[1]. When you feel that you want more information on a topic, that's where to look.

Incidentally, there are a few fibs that appear in this manual; they are used to hide complications (I look at this as something like poetic license). As you become more experienced at using TEX, you'll be able to find them.

TEX is a public domain program that is available for no license fee. It was developed by Donald Knuth at Stanford University as a major project. In the profit-oriented market place, the program would certainly cost many thousands of dollars. The TEX Users Group (TUG) is a nonprofit organization which distributes copies of TEX, this manual, updates software, and gives information about new developments in both hardware and software in its publications TUGboat and TEXniques. Joining this users group is inexpensive; please consider doing so. The address is:

TEX Users Group
P.O. Box 869
Santa Barbara, CA 93102
U.S.A.

[1] Addison-Wesley, Reading,Massachusetts, 1984, ISBN 0-201-13488-9

This manual would not have come into being without the aid of others. In particular the proofreading and suggestions of the following people have been invaluable: Waleed A. Al-Salam (University of Alberta), Debbie L. Alspaugh (University of California), Nelson H. F. Beebe (University of Utah), Barbara Beeton (American Mathematical Society), Anne Brüggemann-Klein (University of Freiburg), Bart Childs (Texas A. & M. University), Mary Coventry (University of Washington), Dimitrios Diamantaras (Temple University), Roberto Dominimanni (Naval Underwater Systems Center), Lincoln Durst (Providence, RI), Victor Eijkhout (University of Nijmegen), Moshe Feder (St. Lawrence University), Josep M. Font (Universidad Barcelona), Jonas de Miranda Gomes (Instituto de Matematica Pura e Aplicada - Brazil), Rob Gross (Boston College), Klaus Hahn (University of Marburg), Anita Hoover (University of Delaware), Jürgen Koslowski (Macalester College), Kees van der Laan (Rijksuniversiteit Groningen), John Lee (Northrop Corporation), Silvio Levy (Princeton University), Robert Messer (Albion College), Emily H. Moore (Grinnell College), Young Park (University of Maryland), Craig Platt (University of Manitoba), Kauko Saarinen (University of Jyväskylä), Jim Wright (Iowa State University), and Dominik Wujastyk (Wellcome Institute for the History of Medicine).

In addition several people have sent me parts or all of their local manuals. I have received a few others on the rebound. In particular Elizabeth Barnhart (TV Guide), Stephan v. Bechtolsheim (Purdue University), Nelson H. F. Beebe (University of Utah) and Leslie Lamport (Digital Equipment Corporation), Marie McPartland-Conn and Laurie Mann (Stratus Computer), Robert Messer (Albion College), Noel Peterson (Library of Congress), Craig Platt (University of Manitoba), Alan Spragens (Stanford Linear Accelerator Center, now of Apple Computers), Christina Thiele (Carleton University), and Daniel M. Zirin (California Institute of Technology) have written various types of lecture notes that have been most helpful.

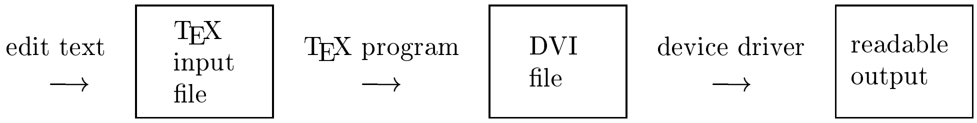
Contents

Section 1
Getting Started

1.1 What TEX is and what TEX isn't

First of all, let's see what steps are necessary to produce a document using TEX. The first step is to type the file that TEX reads. This is usually called the TEX file or the input file, and it can be created using a simple text editor (in fact, if you're using a fancy word processor, you have to be sure that your file is saved in ASCII or nondocument mode without any special control characters). The TEX program then reads your input file and produces what is called a DVI file (DVI stands for DeVice Independent). This file is not readable, at least not by humans. The DVI file is then read by another program (called a device driver) that produces the output that is readable by humans. Why the extra file? The same DVI TEXbook: 23
file can be read by different device drivers to produce output on a dot matrix printer, a laser printer, a screen viewer, or a phototypesetter. Once you have produced a DVI file that gives the right output on, say, a screen viewer, you can be assured that you will get exactly the same output on a laser printer without running the TEX program again.

The process may be thought of as proceeding in the following way:



This means that we don't see our output in its final form when it is being typed at the terminal. But in this case a little patience is amply rewarded, for a large number of symbols not available in most word processing programs become available. In addition, the typesetting is done with more precision, and the input files are easily sent between different computers by electronic mail or on a magnetic medium.

Our focus will be on the first step, that is, creating the TEX input file and then running the TEX program to produce appropriate results. There are two ways of running the TEX program; it can be run in batch mode or interactively. In batch mode you submit your TEX input file to your computer; it then runs the TEX program without further intervention and gives you the result when it is finished. In interactive mode the program can stop and get further input from the user, that is, the user can interact with the program. Using

TEX interactively allows some errors to be corrected by the user, while the TEX program makes the corrections in batch mode as best it can. Interactive is the preferred mode, of course. All personal computer and many mainframe implementations are interactive. On some mainframes, however, the only practical method of running TEX is in batch mode.

1.2 From TEX file to readable output, the big set up

[Note from MD: *This is the only system dependent section in the manual and may be replaced by a local guide. No reference is made to it outside of the section itself. The following local information should be included:*

- *What initial steps, if any, should be taken by the reader to permit the running of TEX and your local device driver(s).*
- *How to run TEX.*
- *How to read the log file.*
- *How to preview and/or print the dvi file.*

The following sample is applicable here at the University of Manitoba. We use a locally written editor (MANTES) on an Amdahl running MVS; I'm fairly certain that it's the worst possible case.]

In this section we'll see how to run TEX at the University of Manitoba. It is assumed that the reader is familiar with MANTES and can create text files using it.

First, there are several things that must be done **one time only**. To start you must do the following (you type in the material that looks like typewriter type):

(1) allocate the files that TEX will use by typing the following lines (while in MANTES):

C: `alloc da=source.tex format=vb,256,6144`

C: `alloc da=dvi format=fb,1024,6144`

(2) Create a file called RUNTEX in your MANTES aggregate containing the following JCL:

```
// JOB ,'RUN TEX'
// EXEC TEXC
//SRC DD DSN=<userid>.SOURCE.TEX,DISP=SHR
//DVI DD DSN=<userid>.DVI,DISP=OLD
```

The name `<userid>` is replaced by your own user id, of course. The use of upper case and the spaces must be followed exactly.

(3) Create a file called PRINTTEX in your MANTES aggregate containing the following JCL:

```
// JOB ,'PRINT TEX'
// EXEC TEXP
//DVIFILE DD DSN=<userid>.DVI,DISP=SHR
```

Once you have completed these three steps, you are ready to run a TEX job. The files you have created will allow you to produce about ten pages of ordinary text.

Here are the steps you use **each time you run a job**.

(1) create a MANTES file containing your TEX input.

(2) save and submit your file using the commands

C: `save f/l to da=source.tex noseq`

C: `submit runtex`

(3) when you get a message saying that your job is finished, enter the command

C: `out <jobname>; list ttyout`

In this command, `<jobname>` is replaced by your user id with a dollar sign appended. This file listing will tell you of any errors that might have occurred. It is an abbreviated version of what is called the "log file"; we will use the term "log file" to refer to the ttyout file produced by TEX.

If you want, you can check on the status of your job while it is executing by using the command

C: `q <jobname>`

When you are finished looking at the log file, the command `end scratch` will throw away the log file while the command `end release` will send the log file to the printer, and it can then be picked up with your TEX output.

(4) when your output from RUNTEX program is error free, TEX will have created a legal DVI file. To print it, use the command

C: `submit printtex nohold`

As in (3), you can check on the status of your job while it is executing.

(5) Pick up your output at the I/O window, sixth floor Engineering building. It usually takes about twenty minutes for the output to be ready. Ask for it by "`<jobname>`".

The files created are large enough for running TEX jobs of about 10 pages. A job of that size will take about one second of CPU time to run through TEX. It will take about 15 seconds of CPU time to print 10 pages on the Xerox 8600 using the current device driver.

You can print your own copy of this manual using the command `%docu tex`. You can also find lots of other useful online information about TEX by using `%texinfo`.

1.3 Let's do it!

So, from our viewpoint, the name of the game is to create the TEX file that produces the right readable output. What does a TEX file look like? It consists of characters from an ordinary terminal, that is, upper and lower case letters, numbers, and the usual punctuation and accent characters (these are the usual ASCII characters). Text, for the most part, is just typed normally. Special instructions usually start with one of a few special symbols such as `#` or `&` (these will be described in more detail later). Here is an example of a TEX input file:

```
Here is my first \TeX\ sentence.
\bye
```

First note that the characters in this example look like typewriter type. We use these characters with all examples that are meant to be typed from the computer terminal. Second, note that the backslash appears three times in the text. We'll soon see that this is one of the special symbols mentioned previously, and it is used very frequently when making TEX documents. Make a file containing this example. Use the TEX program to make a DVI file and a device driver to see what you have produced. If all goes well, you'll have a single page with the following single sentence:

Here is my first TEX sentence.

There will also be a page number at the bottom of the page. If you've gotten this far, congratulations! Once you can produce one TEX document, it's just a matter of time before you can do more complicated ones. Now let's compare what we typed in with what we got out. The straightforward words were just typed in normally, and TEX set them in ordinary type. But the word "TEX", which can't be typed in on a terminal because the letters aren't on the same line, is entered by using a word starting with a backslash, and TEX made the proper interpretation. Most symbols that are not ordinary letters, numbers, or punctuation are typeset by entering a word starting with a backslash. If we look a little closer, we'll note that the word "first" is also changed. The first two letters have been joined together and there isn't a separate dot over the letter "i". This is standard typesetting practice: certain letter combinations are joined up to form what are called *ligatures*. There is actually a good æsthetic reason for this. Compare the first two letters of "first" and "first" to see the difference. We note that `\bye` appears in the input file with no corresponding word in the final copy. This a typesetting instruction that tells TEX that the input is finished. We'll learn about lots of different typesetting instructions as we go along. TEXbook: 4

Let's look at the log file that was created when we ran TEX. It may vary slightly at your site, but should look something like this:

```
1.
2.  This is TeX, MVS Version 2.9 (no format preloaded)
3.  ** File PLAIN.FMT <-- DD=FMTLIB MEM=PLAIN
4.
5.  ** File SRC.TEX <-- DD=SRC
6.  (src.tex [1]
7.  Output written on DVI (1 page, 256 bytes).
8.  Transcript written on TEXLOG.
```

This is the file that will contain any error messages. On line 6, `(src.tex` indicated that TEX has started reading that file. The appearance of `[1]` indicates that page 1 has been processed. If there were errors on page 1, they would be listed at that point.

▷ Exercise 1.1 Add a second sentence to your original TEX file to get:

```
Here is my first \TeX\ sentence.
I was the one who typeset it!
\bye
```

Use TEX and look at your output. Is the second sentence on a new line?

▷ Exercise 1.2   Now add this line to the beginning of your file:

```
\nopagenumbers
```

Guess what will happen when you run the new file through TEX. Now try it and see what happens.

▷ Exercise 1.3   Add three or four more sentences to your file. Use letters, numbers, periods, commas, question marks, and exclamation points, but don't use any other symbols.

▷ Exercise 1.4   Leave a blank line and add some more sentences. You can now get new paragraphs.

We have now seen a major principle concerning the preparation of TEX input files. The placement of the text on your computer terminal does not necessarily correspond to the placement of the text on your output. You can not, for example, add space between words in your output by adding spaces in your input file. Several consecutive spaces and one space will produce exactly the same output. As would be expected, a word at the end of one line will be separated from the first word of the following line. In fact, sometimes when working on a file that will be heavily edited, it is convenient to start each sentence on a separate line. Spaces at the beginning of a line, however, are always ignored.

▷ Exercise 1.5   Add the following sentence as a new paragraph, and then typeset it:

```
Congratulations!  You received a grade of 100% on your latest
examination.
```

The % sign is used for comments in your input file. Everything on a line following this symbol is ignored. Notice that even the space that normally separates the last word on one line from the first word on the next line is lost. Now put a backslash in front of the % sign to correct the sentence.

▷ Exercise 1.6   Add the following sentence as a new paragraph:

```
You owe me $10.00 and it's about time you sent it to me!
```

This will produce an error in your log file (if your implementation of TEX is interactive, that is, sends you messages and waits for answers, just hit the carriage return or enter key when you get the error message). You will get output, but not what you might expect. Look at the log file and see where the error messages are listed. Don't worry about the actual messages. We'll have a lot more to say about errors (including this one) later. Now fix the error by putting a backslash in front of the `$` sign, and typeset the result (there are a small number of characters like the per cent and dollar signs that TEX uses for its own purposes. A table of these characters will be provided shortly).

1.4 TEX has everything under control

We have seen that the backslash has a special role. Any word starting with a backslash will be given a special interpretation when TEX reads it from your input file. Such a word is called a *control sequence*. There are, in fact, two types of control sequences: a *control word* is a backslash followed by letters only (for example, `\TeX`) and a *control symbol* is a backslash followed by a single nonletter (for example, `\$`). Since a space is a nonletter, a backslash followed by a space is a legitimate control symbol. When we want to emphasize that a space is present, we will use a special symbol ␣ to indicate the space, as in the control symbol \␣. This convention is used in **The TEXbook** as well as in this manual. TEXbook: 7–8

When TEX is reading your input file and comes to a backslash followed by a letter, it knows that a control word is being read. So it continues reading the name of the control word until a nonletter is read. So if your file contains

```
I like \TeX!
```

the control word `\TeX` is terminated by the exclamation point. But this presents a problem if you want to have a space after a control word. If you have, for example, the sentence

```
I like \TeX and use it all the time.
```

in your input file, the control word `\TeX` is terminated by the space (which is, of course, a nonletter). But then you won't have a space between the words "TEX" and "and"; inserting more spaces won't help, since TEX doesn't distinguish between one space and several consecutive spaces. But if you put the control symbol \␣ after a control word, you will both terminate the control word and insert a space. It's really easy to forget to put in something like \␣ after a control word. I promise you that you will do it at least once while you're learning to use TEX.

▷ Exercise 1.7 Make an input file that will produce the following paragraph:

I like TEX! Once you get the hang of it, TEX is really easy to use. You just have to master the TEXnical aspects.

Most control words are named so as to give a hint of their uses. You can use `\par` to make a new paragraph, for example, instead of putting in a blank line.

1.5 What TEX won't do

TEX is very good at setting type, but there are things that TEX can't do well. One is making figures. Space can be left to insert figures, but there are no graphic procedures built into the language (at present). Some implementations allow graphic instructions to be inserted using the `\special` control word but these are exceptional and definitely site dependent.

TEX sets type in horizontal straight lines and not in straight lines at other angles. In particular, it is generally not possible to make insertions in "landscape mode", that is, with the text rotated so that the baseline is parallel with the long edge of the paper, or to include text that has a curve for a baseline. Perspective type (gradually increasing or decreasing in height) is not handled well by TEX.

We have seen that there is an "edit, TEX, driver" cycle that is necessary for each different copy of output. This is true even when the output device is a terminal. In particular, it's not possible to type the input file and see the results on the screen immediately without going through the full cycle. Some implementations have both text and graphics displays with reasonably quick turnaround (a few seconds for a single page); as hardware becomes less expensive and processors become faster, we may see improvement.

Section 2
All characters great and small

2.1 Some characters are more special than others

We saw in the last section that most text is entered at the terminal as sentences of ordinary words just as when typing with a typewriter. But we also saw that, in particular, the backslash could be used for at least two different purposes. It can be used for symbols (or combinations of symbols) that don't appear on the keyboard such as typing `\TeX` to get TeX. It can also be used to give TeX special instructions such as typing `\bye` to indicate the end of the input file. In general, a word starting with a backslash will be interpreted by TeX as one requiring special attention. There are several hundred words that TeX knows, and you can define more yourself, and so the backslash is very important. We'll spend a lot of time as we proceed learning some of these words; fortunately we'll only need to use a small number of them most of the time.

There are ten characters which, like the backslash, are used by TeX for special purposes, and we now want to give the complete list. What if we want to use a sentence with one of these special characters in it? With this in mind we'll ask the following questions:

TeXbook: 37–38

(1) What are the different special characters?

(2) How do we use a special character if we really want to typeset it in our text?

Here is a table of each special character, its purpose, and the method of typesetting the special character itself if you need it:

Characters requiring special input

Character	**Purpose**	**Input for literal output**
\	Special symbols and instructions	`$\backslash$`
{	Open group	`$\{$`
}	Close group	`$\}$`
%	Comments	`\%`
&	Tabs and table alignments	`\&`
˜	Unbreakable space	`\~{}`
$	Starting or ending math text	`\$`
ˆ	Math superscripts	`\^{}`
_	Math subscripts	`\_{}`
#	Defining replacement symbols	`\#`

2.2 Typesetting with an accent

Now we're going to start using some of TeX's goodies! So far we've just been using TeX to make our output look attractive, but now we'll start to do things that are difficult or impossible on the typewriter. In particular, we're going to look at accents now. How do you produce an accent when the symbol doesn't appear on the keyboard? Just as with the symbol TeX, it is necessary to enter a word starting with a backslash. For the word "première", as a first example, you need to type in `premi\`ere` (you may have to hunt around to find the "back prime" or "grave" symbol ` on your keyboard, but it's there somewhere[2]). In general, to put an accent on a letter, the appropriate control sequence precedes the letter.

[2] If you have a very old or obscure keyboard and the back prime is *really* not there, you can use `\lq{}` instead. Similarly `\rq{}` can be used for the symbol '. You can think of the symbols as being abbreviations for "left quote" and "right quote." In addition, `\lq\lq{}` and `\rq\rq{}` give the usual quotation marks. But this won't work as a method to produce an accent over the following letter, so you're really better off with a proper keyboard.

Here are some examples:

TEX input	**TEX output**
`` \`a la mode ``	à la mode
`r\'esum\'e`	résumé
`soup\c␣con`	soupçon
`No\"el`	Noël
`na\"\i␣ve`	naïve

We see several principles illustrated by these examples. Most accents are produced by using a control symbol with a similar shape. A few of them are produced by control words containing a single letter. Some care must be used in this case, for a space must be used to terminate the control word. If you have `soup\ccon` in your file, for example, TEX will look for the control word `\ccon`[3]. TEXbook: 52–53

Notice that there is a control word `\i` also. This produces the letter "i" without the dot over it; this allows an accent to be put over the lower part of the letter. There is an analogous control word `\j` that produces a dotless "j" for accenting purposes.

Accents that may be immediately followed by a letter

Name	**TEX input**	**TEX output**
grave	`` \`o ``	ò
acute	`\'o`	ó
circumflex	`\^o`	ô
umlaut/dieresis/trémat	`\"o`	ö
tilde	`\~o`	õ
macron	`\=o`	ō
dot	`\.o`	ȯ

[3] We'll see that there is another method when we look at the grouping concept in Section 4.

Accents requiring an intervening space

Name	TEX input	TEX output
cedilla	`\c o`	ǫ
underdot	`\d o`	ọ
underbar	`\b o`	o̲
háček	`\v o`	ǒ
breve	`\u o`	ŏ
tie	`\t oo`	o͡o
Hungarian umlaut	`\H o`	ő

TEX also allows some letters from languages other than English to be typeset.

Available foreign language symbols

Example	TEX input	TEX output
Ægean, æsthetics	`\AE, \ae`	Æ, æ
Œuvres, hors d'œuvre	`\OE \oe`	Œ, œ
Ångstrom	`\AA, \aa`	Å, å
Øre, København	`\O, \o`	Ø, ø
Łodz, łódka	`\L, \l`	Ł, ł
Nuß	`\ss`	ß
¿Si?	`?‘`	¿
¡Si!	`!‘`	¡
Señor	`\~`	˜
	`{\it\$}`	£

Typeset the sentence in each of the following exercises:

▷ Exercise 2.1 Does Æschylus understand Œdipus?

▷ Exercise 2.2 The smallest internal unit of TEX is about 53.63Å.

▷ Exercise 2.3 They took some honey and plenty of money wrapped up in a £5 note.

▷ Exercise 2.4 Élèves, refusez vos leçons! Jetez vos chaînes!

▷ Exercise 2.5 Zašto tako polako pijete čaj?

▷ Exercise 2.6 Mein Tee ist heiß.

▷ Exercise 2.7 Peut-être qu'il préfère le café glacé.

▷ Exercise 2.8 ¿Por qué no bebes vino blanco? ¡Porque está avinagrado!

▷ Exercise 2.9 Míjn ideeën worden niet beïnvloed.

▷ Exercise 2.10 Can you take a ferry from Öland to Åland?

▷ Exercise 2.11 Türkçe konuşan yeğenler nasillar?

2.3 Dots, dashes, quotes, ...

Typing has always been a compromise: the small number of keys (compared to the number of typeset symbols available) has forced some changes on the typist. When preparing material using TEX, there is no need to be so restricted. In this section we'll look at some differences between typing and using TEX.

There are four types of dashes that are used. The hyphen is used for combining words into one unit as with mother-in-law. The en-dash is used to indicate a sequence of page numbers, years or such things. The em-dash is a grammatical symbol. The minus sign is used for negative numbers. Here they are with their uses: TEXbook: 3–5

Different types of dashes

Name	TEX input	TEX output	Example
hyphen	`-`	-	The space is 3-dimensional.
en-dash	`--`	–	Read pages 3–4.
em-dash	`---`	—	I saw them—there were 3 men alive.
minus sign	`$-$`	$-$	The temperature dropped to -3 degrees.

▷ Exercise 2.12 I entered the room and—horrors—I saw both my father-in-law and my mother-in-law.

▷ Exercise 2.13 The winter of 1484–1485 was one of discontent.

Another difference between typing and using TEX is the use of quotation marks. Opening and closing quotation marks are the same on a typewriter. They are produced in TEX by using the apostrophe or prime key `'` and backprime key `` ` ``. An open quote is produced by ``` `` ``` and the close quote by `''`. Similarly the opening single quote is produced by `` ` `` and the closing single quote by `'`. Notice that there is no need to use the usual typing quotation mark (it normally gives a close quote, but don't count on it). TEXbook: 3

▷ Exercise 2.14 His "thoughtfulness" was impressive.

▷ Exercise 2.15 Frank wondered, "Is this a girl that can't say 'No!'?"

Sometimes ellipses (three dots) are used to indicate a passage of time or missing material. Typing three periods will give three dots very close together. The dots with proper spacing will be produced by having `\dots` in your input file. TEXbook: 173

▷ Exercise 2.16 He thought, "...and this goes on forever, perhaps to the last recorded syllable."

Another problem with dots is that the period after an abbreviation normally has less space after it than does the period at the end of a sentence. There are two ways to solve this problem: the dot can be followed by either `\␣` or `~` to change the spacing. The second alternative will give an unbreakable space; when such a space is between two words, these words must typeset on the same line. The input `Prof.~Knuth` would cause those two words to be typeset on one line. This is desirable for names like Vancouver, B. C. and Mr. Jones so that "Mr." and "Jones" do not end up on separate lines. Notice that no backslash is used with the unbreakable space. TEXbook: 91–92

▷ Exercise 2.17 Have you seen Ms. Jones?

▷ Exercise 2.18 Prof. Smith and Dr. Gold flew from Halifax N. S. to Montréal, P. Q. via Moncton, N. B.

2.4 Different fonts

The most obvious difference between typewritten text and TEX output is undoubtedly the different fonts or types of symbols used. When TEX starts up it has sixteen fonts available. Some of these are used for technical purposes, but even so there are several different typefaces available as can be seen in the following table. A complete list of the sixteen fonts is given in Appendix F of **The TEXbook**. Most of them are used automatically; TEXbook: 427–432 a mathematical subscript, for example, is put in smaller type by TEX with no special user intervention.

To change from the usual (roman) typeface to italic, the control word `\it` is used. To change back to the usual roman typeface, use `\rm`. For example, you could have the following sentence: `I started with roman type, \it switched to italic type, \rm and returned to roman type`. This would give the following: I started with roman type, *switched to italic type,*[4] and returned to roman type.

Typefaces other than italic are also available. Those given below are the ones most commonly used. They are available immediately when TEX starts. A little later we'll see how to use other typefaces that aren't available when TEX starts.

[4] *Notice that the comma and this footnote are in italic type, and this looks a little funny. We'll see that there is another method for changing fonts when we talk about grouping in Section 4.*

Font Samples

Font name	TEX switch sequence	Sample of typeface
Roman	`\rm`	This is roman type.
Boldface	`\bf`	**This is boldface type.**
Italic	`\it`	*This is italic type.*
Slanted	`\sl`	*This is slanted type.*
Typewriter	`\tt`	`This is typewriter type.`
Math symbol[5]	`\cal`	$\mathcal{SOME\ SCRIPT\ LETTERS}$.

The slanted and italic fonts seem quite similar at first. It is easy to see the difference between the letter "a" in each sample. When changing from a slanted or italic font to a roman font, the last letter in the first font will lean towards the first letter of the roman font; this looks cramped, and to compensate there is a little extra space that can be added called the *italic correction*. This is done using the control symbol `\/`. In the following sentence, there is no italic correction after the first sequence of italic letters but there is after the second sequence; see the difference: *If* the italic correction is not used the letters are too close together, but *if* the correction is used, the spacing is better. There is no need for the italic correction when the italic characters are followed by a comma or a period, but your text will definitely look better if you use it before quotation marks or parentheses.

It is possible to use fonts other than those initially defined in TEX when desired (assuming that they are available on your computer system, of course). Different sizes can be used with the aid of the control word `\magstep`. To define your new font you will have to know its name on your computer system. For example, the roman typeface is called "cmr10" on most systems. If you use `\font\bigrm = cmr10 scaled \magstep 1`, you can then use `\bigrm` in exactly the same way as you use `\it` or `\rm`. TEXbook: 13–17

Switching via `\bigrm` will give roman type that is larger than the usual by 20%. `\font\bigbigrm = cmr10 scaled \magstep 2` will define a font that is about 44% larger than the usual roman typeface. The sizes `\magstep 0` to `\magstep 5` are available. On most computers `\magstephalf` is also available; this represents an enlargement of about 9.5%. Here are some samples at various sizes:

[5]This example is cheating since you need to know a little about mathematics input to use it, but I wanted to include it anyway. You'll be able to use these letters after you study the use of mathematical symbols.

Different Font Magnifications

Magnification	**Sample**
`\magstep 0`	Sample text at magstep 0.
`\magstephalf`	Sample text at magstephalf.
`\magstep 1`	Sample text at magstep 1.
`\magstep 2`	Sample text at magstep 2.
`\magstep 3`	Sample text at magstep 3.
`\magstep 4`	Sample text at magstep 4.
`\magstep 5`	Sample text at magstep 5.

It's also possible to use completely new typefaces. These are dependent on system availability, of course. Many systems have a file called CMSS10 which is a font that contains sans serif letters. Using `\font\sf = cmss10` will allow the control word `\sf` to be used in the same manner as `\bf`. Having made this definition, the input

```
\sf Here is a sample of our new Sans Serif font.
```

will produce

Here is a sample of our new Sans Serif font.

▷ Exercise 2.19 What problem might have arisen if we had used `\ss` instead of `\sf` to turn on the Sans Serif font? Hint: if the answer doesn't occur to you at first, think about the German alphabet and it will finally come to you.

▷ Exercise 2.20 Typeset a paragraph of magnified sans serif text.

The extra fonts available may vary from site to site. The ones in the following table are available at most places.

External Fonts Names

CMBSY10	CMBXSL10	CMBXTI10	CMBX10	CMBX12	CMBX5
CMBX6	CMBX7	CMBX8	CMBX9	CMB10	CMCSC10
CMDUNH10	CMEX10	CMFF10	CMFIB8	CMFI10	CMITT10
CMMIB10	CMMI10	CMMI12	CMMI5	CMMI6	CMMI7
CMMI8	CMMI9	CMR10	CMR12	CMR17	CMR5
CMR6	CMR7	CMR8	CMR9	CMSLTT10	CMSL10
CMSL12	CMSL8	CMSL9	CMSSBX10	CMSSDC10	CMSSI10
CMSSI12	CMSSI17	CMSSI8	CMSSI9	CMSSQI8	CMSSQ8
CMSS10	CMSS12	CMSS17	CMSS8	CMSS9	CMSY10
CMSY5	CMSY6	CMSY7	CMSY8	CMSY9	CMTCSC10
CMTEX10	CMTEX8	CMTEX9	CMTI10	CMTI12	CMTI7
CMTI8	CMTI9	CMTT10	CMTT12	CMTT8	CMTT9
CMU10	CMVTT10				

Here's a little background about these names. The first two letters `CM` stands for Computer Modern, the name given to this font family by the designer. The number at the end is the point size: 10 point type is the normal size, 7 point is the normal size for subscripts, and 5 point is the normal size for subscripts of subscripts, 12 point is 20% larger than 10 point, etc. If the letters `CM` are followed by `B`, it is a boldface type. Similarly `R` indicates roman, `I` is for italics, `CSC` is for small caps, `SL` is for slanted, `SS` is for sans serif, `SY` is for symbols, and `TT` is for typewriter type.

▷ Exercise 2.21 Find the fonts available on your system, and print out all the letters and numbers in several of them.

▷ Exercise 2.22 The font `CMR12` is 20% larger than `CMR10`. Also `\magstep 1` magnifies the type by 20%. Take some text and print it using `CMR12` then using `CMR10` scaled by `\magstep 1`. The results are quite different!

Section 3
The shape of things to come

In this section we want to see how to make text have different shapes or sizes. One may use TEX with the default sizes, of course, just as we have been doing so far. We'll now become more creative with our output. In discussing the size of various parts of a page of text, there are several units of measure we can use.

3.1 Units, units, units

TEX can measure length using many types of units. The most common are the inch, the centimetre, the point, the pica. The abbreviations for these units are `in`, `cm`, `pt`, and `pc` respectively. The point is defined by the equation 1 inch = 72.27 points, and the pica by 1 pica = 12 points. Thus a point is rather small and is about the size of a decimal point. Here is a sample to give an idea of the comparative sizes: TEXbook: 57

1 inch:

1 centimetre:

20 points:

1 pica:

Thus points are used to make fine changes; a pica is about the distance between the baselines of two consecutive lines of (unmagnified) normal text. TEX is quite exact about dimensions; internally its smallest unit is less than one four-millionth of an inch. Hence it is the resolution of the output device that will determine the accuracy of the output.

There are two other units that are sometimes useful that have different sizes in different fonts. The `ex` is about the height of a small "x" and the `em` is a little smaller than the width of a capital "M". TEXbook: 60

The shape of the output is generally determined by control words. There are many such words; these allow very fine control of the resulting text. But most of the time only a small number of them are necessary.

3.2 Page shape

The text on a page consists of three basic parts. There is the headline above the main text; this often contains a chapter title, section title, or a page number, and may differ on odd and even pages. Below this is the main text, which includes any footnotes. And finally there is the footline that might contain a page number.

In the examples we have seen so far, the headline has been blank. The footline has contained either a centred number or, if `\nopagenumbers` had been used, has also been blank. We'll have more to say about headlines and footlines later in this section. For the moment let's concentrate on the main text.

To finish one page and start a new one, `\vfill \eject` can be used. The control word `\eject` forces the completion of the current page, while `\vfill` causes any left over vertical space to be gathered at the bottom of the page (you might try leaving out the `\vfill` as an experiment to see how the vertical analogue of justified horizontal lines of text works).

The current horizontal width of the text on the page is described by the control word `\hsize`. It can be changed, say to 4 inches, by `\hsize = 4 in` at any desired time using methods to be described in the next few sections. The value of `\hsize` in effect when the paragraph is completed determines the width of the paragraph. As can be seen in this paragraph, the text width can be changed (in this case to 4 inches) for a single paragraph. Because it represents the current value of the horizontal width, expressions such as `\hsize = 0.75\hsize` can make a new value that is relative to the old one.

The vertical analogue of `\hsize` is `\vsize`, which is the current height of the main text. It can be reset, just like `\hsize`. Thus, `\vsize = 8 in` can be used to change the vertical height of the main text. Note that `\vsize` is the size of the text exclusive of any headline or footline material.

Text can also be shifted across the page. The upper left corner of the text is one inch down and one inch in from the upper left corner of the page. The control words `\hoffset` and `\voffset` are used to shift the text horizontally and vertically. Thus `\hoffset = .75 in` and `\voffset = -.5 in` would shift the text an additional .75 inches to the right and .5 inches up. Most of the time you'll need to set `\hoffset`, `\voffset`, and `\vsize` at the beginning of your document only. TEXbook: 251

Control words for page sizes

Name	TEX Control Word	TEX default value (inches)
horizontal width	`\hsize`	6.5
vertical width	`\vsize`	8.9
horizontal offset[6]	`\hoffset`	0
vertical offset[6]	`\voffset`	0

▷ Exercise 3.1 Enter a paragraph of text that is a few lines long. Take several copies of your paragraph and put `\hsize = 5 in` before one and `\hsize = 10 cm` in front of a second one. Try a few other values for `\hsize`.

▷ Exercise 3.2 Put `\hoffset = .5 in` and `\voffset = 1 in` before the first paragraph of your previous exercise.

▷ Exercise 3.3 Take your previous exercise and put `\vsize = 2 in` before the first paragraph.

In the previous section we saw that it is possible to use fonts of larger size by using the `\magstep` control word. It's also possible to magnify the entire document at once. If `\magnification = \magstep 1` appears at the top of your document, then all of your output is magnified by 20%. The other values of `\magstep` can be used, too. **It should be emphasized that `\magnification` can only be used before even a single character is typeset.** This magnification creates a problem in units; if the document is to be magnified by 20% and `\hsize = 5 in` appears in the TEX input file, should the final output have an `\hsize` of 5 inches or should it be magnified by 20% to 6 inches? Unless told otherwise, all the dimensions will be magnified so that, in this case, `\hsize` would be 6 inches in the final output. This means that all the dimensions will increase uniformly when `\magnification` is used. But for some special situations it might not be desirable for this to happen; for example, you might want to leave a space of exactly 3 inches to insert a figure. In this case any unit can be modified by `true` so that `\hsize = 5 true in` will always set the line width to 5 inches, regardless of magnification. TEXbook: 59–60

[6] The text is initially 1 inch down and 1 inch in from upper left corner of the page.

▷ Exercise 3.4 Put `\magnification = \magstep 1` as the first line in one of your input files and see how the output changes.

3.3 Paragraph shape

When TEX is reading the text to be typeset from your input file, it reads in a paragraph at a time and then typesets it. This means that there is a lot of control over paragraph shapes, but also that some care must be taken. We've already seen that `\hsize` can be used to control the width of a single paragraph. But suppose the following appeared as part of your input file:

```
\hsize = 5 in
Four score and seven years ...
⋮
... from this earth.
\hsize = 6.5 in
```

What is the width of the paragraph? The `\hsize` was changed at the start of the paragraph and again at the end. Since the paragraph was not completed (by putting in a blank line or `\par`) until after the second change, the paragraph would be typeset with a width of 6.5 inches. However, if a blank line were inserted before `\hsize = 6.5 in`, then the paragraph would be typeset with a width of 5 inches. So, in general, when a paragraph is set, the values of the parameters that are in effect when the paragraph is completed are the ones that are used.

Here is a table with some paragraph parameters:

Some paragraph shape parameters

Function	TEX Control Word	TEX default
width	`\hsize`	6.5 inches
indentation on first line	`\parindent`	20 points
distance between lines	`\baselineskip`	12 points
distance between paragraphs	`\parskip`	0 points

The control word `\noindent` may be used at the beginning of a paragraph to avoid the automatic indentation on the first line. This control word will only affect the paragraph being set when it is invoked (of course `\parindent = 0 pt` will cause no indentation for all paragraphs).

> A more flexible way to control the width of a paragraph is to use `\rightskip` and `\leftskip`. Setting `\leftskip = 20 pt` causes the left margin of the paragraph to be moved in an extra twenty points. Negative values may be assigned to `\leftskip` to move the left margin out. The control word `\rightskip` does the same to the right side of the paragraph. The single control word `\narrower` is the equivalent of setting both `\leftskip` and `\rightskip` equal to `\parindent`. This is quite useful for long quotations, and this paragraph is an example of its use. As with `\hsize`, the value of `\leftskip` and `\rightskip` in effect when the paragraph is completed is the one which will apply to the whole paragraph. TEXbook: 100

▷ Exercise 3.5 Make two paragraphs with the following specifications: the left margin of both paragraphs is indented by 1.5 inches, the right margin of the first paragraph is indented by 0.75 inches, and the right margin of the second paragraph is indented by 1.75 inches.

Lines can be made with different lengths within one paragraph by using `\hangindent` and `\hangafter`. The amount of the indentation is determined by value of `\hangindent`. If `\hangindent` is positive, the indentation is made from the left, and if it is negative it is made from the right. The lines on which the indentation occurs is controlled by `\hangafter`. If `\hangafter` is positive then it determines the number of lines at full width before the indentation starts. Thus if `\hangindent = 1.75 in` and `\hangafter = 6`, then the first six lines will be at full width and the rest will be indented by 1.75 inches from the left. On the other hand if `\hangindent = -1.75 in` and `\hangafter = -6`, then the first six lines will be indented by 1.75 inches from the right and the rest will be at full width. TEX resets to the default values `\hangindent = 0 pt` and `\hangafter = 1` after each paragraph. These control words are useful for paragraphs with "hanging indents" and for flowing a paragraph around space reserved for a figure. The control word `\hang` at the beginning of the paragraph will cause the first line to be of full width (`\hangafter=1`) and the rest of the paragraph to be indented by the current value of `\parindent`. But you do have to use `\noindent` if you want the first line to extend all the way to the left margin. TEXbook: 355 TEXbook: 102

Here is the previous paragraph repeated with `\hangafter = -6` and `\hangindent = -1.75 in`.

Lines can be made with different lengths within one paragraph by using `\hangindent` and `\hangafter`. The amount of the indentation is determined by value of `\hangindent`. If `\hangindent` is positive, the indentation is made from the left, and if it is negative it is made from the right. The lines on which the indentation occurs is controlled by `\hangafter`. If `\hangafter` is positive then it determines the number of lines at full width before the indentation starts. Thus if `\hangindent = 1.75 in` and `\hangafter = 6`, then the first six lines will be at full width and the rest will be indented by 1.75 inches from the left. On the other hand if `\hangindent = -1.75 in` and `\hangafter = -6`, then the first six lines will be indented by 1.75 inches from the right and the rest will be at full width. TEX resets to the default values `\hangindent = 0 pt` and `\hangafter = 1` after each paragraph. These control words are useful for paragraphs with "hanging indents" and for flowing a paragraph around space reserved for a figure. The control word `\hang` at the beginning of the paragraph will cause the first line to be of full width (`\hangafter=1`) and the rest of the paragraph to be indented by the current value of `\parindent`. But you do have to use `\noindent` if you want the first line to extend all the way to the left margin.

TEXbook: 355
TEXbook: 102

The control word `\parshape` can be used to make paragraphs with a greater variety of shapes.

TEXbook: 101

Another useful control word for setting paragraphs is `\item`. It can be used to make various types of itemized lists, and is invoked using the pattern `\item{...}`. This causes the next paragraph to be formed with every line indented by `\parindent` and, in addition, the first line labeled on the left by whatever is between the braces. It is usually used with `\parskip = 0 pt`, since that control word determines the vertical space between the different items. The control word `\itemitem` is the same as `\item` except that the indentation is twice as far, that is, twice the value of `\parindent`. Here is an example:

TEXbook: 102

```
\parskip = 0pt \parindent = 30 pt
\noindent
Answer all the following questions:
\item{(1)} What is question 1?
\item{(2)} What is question 2?
\item{(3)} What is question 3?
\itemitem{(3a)} What is question 3a?
\itemitem{(3b)} What is question 3b?
```

will produce

Answer all the following questions:
(1) What is question 1?
(2) What is question 2?
(3) What is question 3?
  (3a) What is question 3a?
  (3b) What is question 3b?

▹ Exercise 3.6 Make a paragraph several lines long and use it with the `\item` control word to see the "hanging indent." Now take the same paragraph and use it with different values of `\parindent` and `\hsize`.

Now let's see how to put space between paragraphs. The control word `\parskip` is used to determine how much space is normally left between paragraphs. So if you put `\parskip = 12 pt` at the beginning of your TEX source file, there will be 12 points between paragraphs unless other instructions are given. The control word `\vskip` can be used to insert extra vertical space between paragraphs. If `\vskip 1 in` or `\vskip 20 pt` appears between two paragraphs, then the extra space is inserted.

There are a couple of peculiarities of `\vskip` that seem quite strange at first. If you have `\vskip 3 in` and the skip starts two inches from the bottom of the page, the rest of the page is skipped, but the extra one inch is *not* skipped at the top of the next page. In other words, `\vskip` **will not insert space across page boundaries**. In fact, `\vskip 1 in` will have no effect at all if it happens to appear at the top of a page! For many types of vertical spacing this is quite appropriate. The space before a section heading, for example, should not continue across page boundaries.

A similar phenomenon occurs at the beginning of your document. If, for example, you want a title page with the title about half way down the page, you can not insert the space at the top of the page using `\vskip`.

But what if you really want some blank space at the top of a page? You could start the page with `\␣` but this in effect typesets a one line paragraph containing a blank. So while nothing is typeset, the extra space due to the values of `\baselineskip` and `\parskip` will add extra space. An easier method is to use `\vglue` instead of `\vskip` to get the desired result. Thus `\vglue 1 in` will leave one inch of blank space at the top of the page. TEXbook: 352

We can note in passing that there is another more general method to put material at the top of a page using the control words `\topinsert` and `\endinsert`. If `\topinsert` ... `\endinsert` is used within a page, the material between `\topinsert` and `\endinsert` will appear at the top of the page, if possible. For example:

```
\topinsert
\vskip 1 in
\centerline{Figure 1}
\endinsert
```

is useful for inserting figures of a prescribed size. TEXbook: 115

There are also some special control words for making small vertical skips. They are `\smallskip`, `\medskip`, and `\bigskip`. Here is the size of each one:

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

3.4 Line shape

For most material TEX does a good job of breaking up a paragraph into lines. But sometimes it's necessary to add further instructions. It's possible to force a new line by inserting `\hfill \break` in your input file. It's also possible to put some text on a line by itself using the control word `\line{...}`; then the material between the braces will be restricted to one line (although the result will be spread out over the whole line and may be horrible). The control words `\leftline{...}`, `\rightline{...}`, and `\centerline{...}` will, respectively, set the material between the braces on a single line on the left margin, on the right margin, or centred. Thus

```
\leftline{I'm over on the left.}
\centerline{I'm in the centre.}
\rightline{I'm on the right.}
\line{I just seem to be spread out all over the place.}
```

produces the four lines:

I'm over on the left.

I'm in the centre.

I'm on the right.

I just seem to be spread out all over the place.

Other types of spacing can be obtained by using the control word `\hfil`. This causes all the extra space in the line to be accumulated at the position where `\hfil` appears. Thus if we alter our last example to `\line{I just seem to be spread out \hfil all over the place.}` we will get

I just seem to be spread out all over the place.

If we have more than one `\hfil`, they will divide any extra space among themselves equally. Hence `\line{left text \hfil centre text \hfil right text.}` will produce

left text centre text right text.

▷ Exercise 3.7 Typeset the following line:
left end left tackle left guard centre right guard right tackle right end

▷ Exercise 3.8 Typeset the following line with twice as much space between "left" and "right-centre" as between "right-centre" and "right":
left right-centre right

It's possible to move horizontally using `\hskip` in a manner analogous with `\vskip`.

▷ Exercise 3.9 What happens with the following input:
`\line{\hskip 1 in ONE \hfil TWO \hfil THREE}`

The right justification can be canceled by using the control word `\raggedright`.

3.5 Footnotes

The general pattern to make footnotes using TEX is `\footnote{...}{...}`.

The footnote mark goes in between the first set of braces. Some available marks are `\dag` (†), `\ddag` (‡), `\S` (§), and `\P` (¶). The text of the footnote goes between the second set of braces. The use of numbers as marks is a little less straightforward. The footnote[21] at the bottom of the page was produced by using `\footnote{${}^{21}$}{This is the footnote at the bottom of the page.}` after the word "footnote" in the text. This construction is somewhat complicated; we'll see why it's this way when we know a little more about typesetting mathematics. But for the moment we can look at it as a way of getting the job done. You may want to use `\rm` within the footnote to ensure that the right font appears. Usually you will not want a space between the text and the control word `footnote`. TEXbook: 117

[21] This is the footnote at the bottom of the page.

▹ Exercise 3.10 Make up a page with a relatively long footnote spanning several lines.

▹ Exercise 3.11 Make up a page with two different footnotes on it.

3.6 Headlines and Footlines

The lines for title and page numbers that go above and below the main text are produced by using `\headline={...}` and `\footline={...}`. TEXbook: 252–253

The principle is the same as using the control word `\line{...}` within the usual text on the page. A helpful control word is `\pageno` which represents the current page number. Thus `\headline={\hfil \tenrm Page \the\pageno}` would cause the page number to appear in the upper right corner preceded by the word "Page" (now look at the upper right corner of this page). It is safer to explicitly name the font that you want to use (in this case `\tenrm` to use the 10 point roman font), since you are never guaranteed that a particular font will be in use when the headline or footline is set. The control word `\the` takes the internal value of the next word if it is an appropriate control word and prints it as text. You can also use the control word `\folio` instead of `\the \pageno`. The difference is that `\folio` will give roman numerals when `\pageno` is negative.

You can also assign values to `\pageno` if you want your document to use a different sequence of page numbers from the usual. Roman numerals can be produced by using negative numbers; `\pageno=-1` at the beginning of a document will cause page numbers to be in roman numerals. TEXbook: 252

Different headlines can be produced for even and odd pages in the following manner:

```
\headline={\ifodd \pageno {...}\else {...}\fi}
```

where the material between the first set of braces is for the right-hand pages and the material between the second set of braces is for the left-hand pages.

▹ Exercise 3.12 Change the footline so that the page number is centred with an en-dash on either side.

3.7 Overfull and underfull boxes

One of the most frustrating experiences for the new TEX user is the occurrence of overfull and underfull boxes. They are duly noted in the log file and are also written to the screen when TEX is being run interactively. Overfull boxes are also marked on the output by a slug (a large filled-in black rectangle that looks like this: ▮) in the right margin. Various obscene terms are applied to this slug. It appears even though there is nothing wrong with the TEX input. So why is the slug there and what can be done about it?

A good way to visualize the way TEX organizes a page is to think of the printed material as being put into boxes. There are two types of boxes: *hboxes* and *vboxes*. Most of the time these correspond to the organization of horizontal text into lines and vertical paragraphs into pages. In particular, it is the spacing of the words in a hbox corresponding to a line of text that causes the slug to appear.

Recall that TEX reads in the complete paragraph before deciding how to break it up into lines. This is better than working a line at a time, since a slight improvement in one line might cause catastrophic changes farther down in the paragraph. When words are gathered together to form a line, space is added between the words to justify the right margin. Very large spaces between words is obviously undesirable; the badness of the line is a measure of how badly the words are spaced. An underfull hbox means that there is too much space between words. More specifically, the badness of any line is a number between 0 (perfect) and 10000 (horrible). There is a parameter called `\hbadness` whose default value is 1000. Any line whose badness is greater than `\hbadness` is reported as an underfull hbox. If the value of `\hbadness` is increased, then fewer underfull hboxes will be reported. In fact `\hbadness = 10000` will suppress the reporting of underfull hboxes altogether. Similarly, if the words must be squeezed into a line so that the spaces are very small or even so the words extend a little into the right margin, then an overfull hbox results.

In a similar way, sometimes TEX allows a line to be slightly longer than `\hsize` in order to achieve a more balanced appearance. The control word `\tolerance` determines when this happens. If the badness of a line is greater than `\tolerance`, TEX will make the line longer by adding a new word at the end of the line, even though it might exceed the value of `\hsize`. A line which is only slightly longer is set without being reported. The control word `\hfuzz` determines how much excess is allowed; the default is `\hfuzz = 0.1 pt`. A line that is more than slightly longer than `\hsize` is obviously a serious problem; TEX puts a slug in the margin to make its point forcefully. It's possible to avoid overfull hboxes altogether by increasing the value of `\tolerance`. With `\tolerance = 10000` there will never be overfull boxes or slugs. The default is `\tolerance = 200`. TEXbook: 29

The width of the slug is determined by the control word `\overfullrule`. Including `\overfullrule = 0 pt` in your file will delete any further printing of the slugs. The overfull boxes will still be there, of course, and they will probably be harder to spot.

So we see why overfull boxes and underfull boxes are reported; we can also change the reporting by changing the values of `\badness`, `\hfuzz`, and `\tolerance`. In addition, a small value of `\hsize` obviously makes it more difficult to set lines and causes more overfull and underfull hboxes to be reported. These are warnings that you ignore at your own peril!

Adding new possibilities of hyphenation will sometimes eliminate an overfull box. TEX has automatic hyphenation and usually finds good places to hyphenate a word; however, it's possible to add hyphenations to let lines break at new places. For example, the automatic hyphenation will never put a hyphen in the word "database". If you type in `data\-base`, it will allow a hyphen to be inserted after the second letter "a" in the word. More generally, if you put `\hyphenation{data-base}` in the beginning of your input file, then all occurrences of the word "database" will allow hyphenation after the letter "a". The log file will show the possible hyphenations on the line containing the overfull or underfull box. Sometimes the best solution to an overfull or underfull hbox is a little judicious editing of the original document. TEXbook: 28

Our discussion has involved the setting of type into lines, that is, the horizontal page structure. There are several vertical analogues. Overfull and underfull hboxes indicate how well words are gathered into lines. Similarly, overfull and underfull vboxes are reported when paragraphs are gathered to form pages. A large table that can't be broken in the middle, for example, can produce an underfull vbox that is reported in the log file when the page being typeset is completed. The control word `\vbadness` works for the vertical placement of text in the same way as `\hbadness` works for horizontal text.

▹ Exercise 3.13 Take a few paragraphs and print them using various (small) values of `\hsize` to see what kind of overfull boxes result. Repeat with various values of `\hbadness`, `\hfuzz`, and `\tolerance`.

Section 4
{Groups, {Groups, {and More Groups}}}

The concept of gathering text into groups allows TEX input files to be greatly simplified. A new group is started by the character `{` and terminated by the character `}`. Changes made within a group will lose their effect when the group terminates. So, for example if `{\bf three boldface words}` appears in your text, the opening brace starts the group, the `\bf` control word changes to a boldface font, and the closing brace finishes up the group. Upon completion of the group the font being used is the one in effect before the group started. This is the (easier) way of having a few words in a different font. It's also possible to have groups nested within groups.

As another example, size changes can be made in the text that are only temporary. For example

```
{
\hsize = 4 in
\parindent = 0 pt
\leftskip = 1 in
will produce a paragraph that is four
⋮
(this is an easy mistake to make).
\par
}
```

will produce a paragraph that is four inches wide with the text offset into the paragraph by one inch regardless of the settings in effect before the start of the group. This paragraph is set with those values. After the end of the group, the old settings are in effect again. Note that it is necessary to include `\par` or to use a blank line before the closing brace to end the paragraph, since otherwise the group will end and TEX will go back to the old parameters before the paragraph is actually typeset (this is an easy mistake to make).

When line-spacing control words (like `\centerline`) act on text following it in braces, that text is implicitly in a group. Thus `\centerline{\bf A bold title}` will produce a centred boldface line, and the text following that line will be in whatever font was in effect before the `\centerline` was invoked.

The empty group `{}` is useful. One use allows accents to be typeset with no accompanying letter. For example, `\~{}` will print a tilde with no letter under it. It can also be used to stop TEX from eating up consecutive spaces. Hence `I use \TeX{} all the time` will leave a space after "TEX" in the output. This is an alternative to using `\␣` as we did in Section 1. TEXbook: 19–21

Grouping can also be used to avoid spaces in the middle of a word when including accents. Either `soup\c␣con` or `soup\c{c}on` will produce the word soupçon.

▷ Exercise 4.1 Change the dimensions of one paragraph on a page using the grouping idea.

▷ Exercise 4.2 Mathematicians sometime use the word "iff" as an abbreviation for "if and only if". In this case it looks better if the first and second letter "f" are *not* joined as a ligature. How do you do this (there are several solutions)?

It's really easy to forget to match braces properly. The effect can be dramatic; if you get output that suddenly changes to an italic font for the rest of the document, a mismatched brace is probably the cause. If you have an extra `{` TEX will give a message in the log file: `(\end occurred inside a group at level 1)`. An extra `}` will result in the message `! Too many }'s.`

Here's a little hint to help you keep track of the braces in more complicated groups: put the opening brace on a line by itself and do the same for the closing brace. If there are braces nested within the original ones, put them on separate lines also, but indent them a few spaces. The text within the nested braces can also be indented since TEX ignores all spaces at the beginning of a line. The matching braces will then stand out when you look at your TEX source file. In fact, if your editor is smart enough, you can create the two lines with the braces first and then insert the appropriate material within them with automatic indenting.

▷ Exercise 4.3 In section 2 we changed fonts the following method: `I started with roman type, \it switched to italic type, \rm and returned to roman type`. Get the same result using the idea of grouping.

Section 5
No math anxiety here!

TEX is at its best when typesetting mathematics. The conventions for doing this are many and complex, and the ability of TEX to take them into account makes the production of high quality, attractive mathematical output possible. If you plan to produce papers with mathematical symbols in them, this section will give you all the basics necessary for creating beautiful output in almost all circumstances; TEX may be used without any mathematics, of course, and if this is your goal, then the following two subsections are probably sufficient for your needs.

5.1 Lots of new symbols

Mathematical text is inserted into normal text in two possible ways: it can be *in-line*, that is, as part of the usual lines of text. It can also be *displayed*, that is, centred in a gap between the usual text. The results in the spacing and placement of symbols can be quite different in each case. The in-line equation $\sum_{k=1}^\infty \frac{1}{k^2} = \frac{\pi^2}{6}$ is easily seen to be different from the same equation when displayed:

$$\sum_{k=1}^\infty \frac{1}{k^2} = \frac{\pi^2}{6}.$$

Since the spacing and the fonts used for mathematics are quite different from those of ordinary text, TEX needs to be told when mathematics rather than normal text is being typeset. This is done using the `$` symbol. More specifically, mathematics is typeset in-line by surrounding the symbols to be entered by single dollar signs: `$...$`, and is typeset displayed by surrounding the symbols to be entered by double dollar signs: `$$...$$`. Hence `$x = y+1$` will give $x = y + 1$ in-line while `$$x = y+1.$$` will display

$$x = y + 1.$$

The spacing for both in-line and displayed mathematics is completely controlled by TEX. Adding spaces to your input has no effect at all. What if you need a space or some text in the middle of some mathematics? You can insert text by inserting it into an hbox (don't worry about the definition of an hbox for now): `\hbox{...}`. This is particularly useful for displayed mathematics. Hence "$x = y + 1$ whenever $y = x - 1$" can be typeset using `$x=y+1 \hbox{ whenever }y=x-1$`. Note the spaces on either side of the word within the braces. Usually it isn't necessary to insert spaces into mathematical text, but here are the control sequences that do the job. TeXbook: 167

Adding space to mathematical text

Name	Control Sequence	←Size→
Double quad	`\qquad`	
Quad	`\quad`	
Space	`\␣`	
Thick space	`\;`	
Medium space	`\>`	
Thin space	`\,`	
Negative thin space	`\!`	

If you look closely at the negative thin space, you'll notice that, unlike the other entries, the two arms overlap. This is because the negative space is one in the opposite direction, that is, while the other control sequences increase the amount of space between two symbols being typeset, the negative thin space decreases the space between them, even if it causes them to overlap.

▷ Exercise 5.1 Typeset the following: $C(n,r) = n!/(r!\,(n-r)!)$. Note the spacing in the denominator.

You shouldn't have any blank lines between the dollar signs delimiting the mathematical text. TeX assumes that all the mathematical text being typeset is in one paragraph, and a blank line starts a new paragraph; consequently, this will generate an error message. This turns out to be useful, for one of the easiest errors to make is to forget to put in the trailing dollar sign(s) after the mathematical input (I promise that you'll do it at least once while learning TeX); if TeX allowed more than one paragraph to be between the dollars signs, then one omitted trailing dollar sign might cause the rest of the document to be typeset as mathematics.

Most mathematical text is entered in exactly the same way for in-line typesetting as for displayed typesetting (except for the surrounding dollar signs, of course). The exceptions, such as aligning multiline displays and placing equation numbers at the left or right margin will be discussed in the last part of this section.

Many new symbols can appear when typesetting mathematics. Most of the ones that actually appear on the keyboard can be used directly. The symbols `+ - / * = ' | < > (` and `)` are all entered directly. Here they are as mathematics: $+ - /* = {}' \mid < > (\,)$.

▷ Exercise 5.2 Typeset the equation $a + b = c - d = xy = w/z$ as in-line and displayed mathematical text.

▷ Exercise 5.3 Typeset the equation $(fg)' = f'g + fg'$ as in-line and displayed mathematical text.

Many other symbols, as you would expect, are predefined control words. All Greek letters are available. Here is a table of them: TEXbook: 434

Greek letters

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

▷ Exercise 5.4 Typeset $\alpha\beta = \gamma + \delta$ as in-line and displayed mathematical text.

▷ Exercise 5.5 Typeset $\Gamma(n) = (n-1)!$ as in-line and displayed mathematical text.

Sometimes accents are put above or below symbols. The control words used for accents in mathematics are different from those used for normal text. The normal text control words may not be used for mathematics and vice-versa. TEXbook: 135–136

Mathematical accents

$\hat o$	`\hat o`	$\check o$	`\check o`	$\tilde o$	`\tilde o`
$\acute o$	`\acute o`	$\grave o$	`\grave o`	$\dot o$	`\dot o`
$\ddot o$	`\ddot o`	$\breve o$	`\breve o`	$\bar o$	`\bar o`
$\vec o$	`\vec o`	$\widehat{abc}$	`\widehat {abc}`	$\widetilde{abc}$	`\widetilde {abc}`

Binary operators combine two mathematical objects to get another object. Ordinary addition and multiplication, for example, combine two numbers to get another number, and so they are binary operators. When a binary operator such as $+$ or $\times$ is typeset, a little extra space is put around it. Here is a list of some of the available binary operators: TEXbook: 436

Binary operators

$\cdot$	`\cdot`	$\times$	`\times`	$\ast$	`\ast`	$\star$	`\star`
$\circ$	`\circ`	$\bullet$	`\bullet`	$\div$	`\div`	$\diamond$	`\diamond`
$\cap$	`\cap`	$\cup$	`\cup`	$\vee$	`\vee`	$\wedge$	`\wedge`
$\oplus$	`\oplus`	$\ominus$	`\ominus`	$\otimes$	`\otimes`	$\odot$	`\odot`

Ellipses are commonly used with binary operators. The control word `\cdots` will raise the dots so that they are level with the binary operator. Thus `$a + \cdots + z$` will produce $a + \cdots + z$. The control word `\ldots` will put the dots on the baseline, and so `$1\ldots n$` produces $1\ldots n$.

▷ Exercise 5.6 Typeset: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

▷ Exercise 5.7 Typeset: $2 + 4 + 6 + \cdots + 2n = n(n + 1)$.

A relation indicates a property of two mathematical objects. We already know how to show two objects equal, or how to show one number less than or greater than another number (since these are symbols on most terminal keyboards). To negate a relation, the control word `\not` is put in front of the relation. Here are some relations: TEXbook: 436

Relations

$\leq$	`\leq`	$\not\leq$	`\not \leq`	$\geq$	`\geq`	$\not\geq$	`\not \geq`
$\equiv$	`\equiv`	$\not\equiv$	`\not \equiv`	$\sim$	`\sim`	$\not\sim$	`\not \sim`
$\simeq$	`\simeq`	$\not\simeq$	`\not \simeq`	$\approx$	`\approx`	$\not\approx$	`\not \approx`
$\subset$	`\subset`	$\subseteq$	`\subseteq`	$\supset$	`\supset`	$\supseteq$	`\supseteq`
$\in$	`\in`	$\ni$	`\ni`	$\parallel$	`\parallel`	$\perp$	`\perp`

▹ Exercise 5.8 Typeset: $\vec{x} \cdot \vec{y} = 0$ if and only if $\vec{x} \perp \vec{y}$.

▹ Exercise 5.9 Typeset: $\vec{x} \cdot \vec{y} \neq 0$ if and only if $\vec{x} \not\perp \vec{y}$.

Here are some other available mathematical symbols: TEXbook: 435–438

Miscellaneous symbols

$\aleph$	`\aleph`	ℓ	`\ell`	$\Re$	`\Re`	$\Im$	`\Im`
∂	`\partial`	∞	`\infty`	$\|$	`|`	$\angle$	`\angle`
∇	`\nabla`	$\backslash$	`\backslash`	$\forall$	`\forall`	$\exists$	`\exists`
$\neg$	`\neg`	$\flat$	`\flat`	$\sharp$	`\sharp`	$\natural$	`\natural`

▹ Exercise 5.10 Typeset: $(\forall x \in \Re)(\exists y \in \Re)\ y > x$.

5.2 Fractions

There are two methods of typesetting a fraction: it can be typeset either in the form 1/2 or in the form $\frac{1}{2}$. The first case is just entered with no special control sequences, that is, `$1/2$`. The second case uses the control word `\over` and the following pattern: `{<numerator> \over <denominator>}`. Hence `$${a+b \over c+d}.$$` gives TEXbook: 139–140

$$\frac{a+b}{c+d}.$$

▷ Exercise 5.11 Typeset the following: ${a+b \over c}$ ${a \over b+c}$ ${1 \over a+b+c} \ne {1 \over a} + {1 \over b} + {1 \over c}$.

▷ Exercise 5.12 Typeset: What are the points where ${\partial \over \partial x} f(x,y) = {\partial \over \partial y} f(x,y) = 0$?

5.3 Subscripts and superscripts

Subscripts and superscripts are particularly easy to enter using TEX. The characters _ and ^ are used to indicate that the next character is a subscript or a superscript. Thus `$x^2$` gives x^2 and `$x_2$` gives x_2. To get several characters as a subscript or superscript, they are grouped together within braces. Hence we can use `$x^{21}$` to get x^{21} and `$x_{21}$` to get x_{21}. Notice that the superscripts and subscripts are automatically typeset in a smaller type size. The situation is only slightly more complicated for a second layer of subscripts or superscripts. You can *not* use `$x_2_3$` since this could have two possible interpretations, namely, `$x_{2_3}$` or `${x_2}_3$`; this gives two different results: x_{2_3} and ${x_2}_3$, the first of which is the usual mathematical subscript notation. Thus you must put in the complete braces to describe multiple layers of subscripts and superscripts. They may be done to any level. TEXbook: 128–130

To use both subscripts and superscripts on one symbol, you use both the _ and ^ in either order. Thus either `$x_2^1$` or `$x^1_2$` will give x_2^1.

▷ Exercise 5.13 Typeset each of the following: e^x e^{-x} $e^{i\pi} + 1 = 0$ x_0 x_0^2 ${x_0}^2$ 2^{x^x}.

▷ Exercise 5.14 Typeset: $\nabla^2 f(x,y) = {\partial^2 f \over \partial x^2} + {\partial^2 f \over \partial y^2}$.

A similar method is used for summations and integrals. The input of `$\sum_{k=1}^n k^2$` will give $\sum_{k=1}^n k^2$, and `$\int_0^x f(t) dt$` will give $\int_0^x f(t) dt$. TEXbook: 144–145

Another use of this type of input is for expressions involving limits. You can use `$\lim_{n\to \infty}({n+1 \over n})^n = e$` to get $\lim_{n\to\infty}({n+1 \over n})^n = e$.

▷ Exercise 5.15 Typeset the following expression: $\lim_{x\to 0}(1+x)^{1 \over x} = e$.

▷ Exercise 5.16 Typeset: The cardinality of $(-\infty, \infty)$ is $\aleph_1$.

▷ Exercise 5.17 Typeset: $\lim_{x\to 0^+} x^x = 1$.

Here's a hint to make integrals look a little nicer: look at the difference between $\int_0^x f(t)dt$ and $\int_0^x f(t)\,dt$. In the second case there is a little extra space after $f(t)$, and it looks nicer; `\,` was used to add the additional space.

▷ Exercise 5.18 Typeset the following integral: $\int_0^1 3x^2\,dx = 1$.

5.4 Roots, square and otherwise

To typeset a square root it is only necessary to use the construction `\sqrt{...}`. Hence `$\sqrt{x^2+y^2}$` will give $\sqrt{x^2+y^2}$. Notice that TEX takes care of the placement of symbols and the height and length of the radical. To make cube or other roots, the control words `\root` and `\of` are used. You get $\sqrt[n]{1+x^n}$ from the input `$\root n \of {1+x^n}$`. TEXbook: 130–131

A possible alternative is to use the control word `\surd`; the input `$\surd 2$` will produce $\surd 2$.

▷ Exercise 5.19 Typeset the following: $\sqrt{2} \quad \sqrt{\frac{x+y}{x-y}} \quad \sqrt[3]{10} \quad e^{\sqrt{x}}$.

▷ Exercise 5.20 Typeset: $\|x\| = \sqrt{x\cdot x}$.

▷ Exercise 5.21 Typeset: $\phi(t) = \frac{1}{\sqrt{2\pi}}\int_0^t e^{-x^2/2}\,dx$.

5.5 Lines, above and below

Use the constructions `\overline{...}` and `\underline{...}` to put lines above or below mathematical expressions. Hence `$\overline{x+y}=\overline x + \overline y$` gives $\overline{x+y} = \overline x + \overline y$. But notice that the lines over the letters are at different heights, and so some care is necessary. The use of `\overline{\strut x}` will raise the height of the line over x. TEXbook: 130–131

To underline non-mathematical text, use `\underbar{...}`.

▷ Exercise 5.22 Typeset the following: $\underline{x}$ $\overline{y}$ $\underline{\overline{x+y}}$.

5.6 Delimiters large and small

The most commonly used mathematical delimiters are brackets, braces, and parentheses. As we have seen, they may be produced by using `[ ] \{ \} ( )` to get $[\,]\,\{\,\}\,(\,)$. Sometimes larger delimiters increase the clarity of mathematical expressions, as in

$$\bigl(a \times (b+c)\bigr)\bigl((a \times b)+c\bigr).$$

To make larger left delimiters the control words `\bigl`, `\Bigl`, `\biggl`, and `\Biggl` are used in front of the delimiter; similarly, `\bigr`, `\Bigr`, `\biggr`, and `\Biggr` are used for the right TeXbook: 145–147 delimiters. Hence `$\Bigl[$` and `$\Bigr]$` will produce $\Bigl[$ and $\Bigr]$.

Here is a table to compare the size of some of the delimiters.

Delimiters of various sizes

$\{$	`\{`	$\}$	`\}`	$($	`(`	$)$	`)`
$\bigl\{$	`\bigl\{`	$\bigr\}$	`\bigr\}`	$\bigl($	`\bigl(`	$\bigr)$	`\bigr)`
$\Bigl\{$	`\Bigl\{`	$\Bigr\}$	`\Bigr\}`	$\Bigl($	`\Bigl(`	$\Bigr)$	`\Bigr)`
$\biggl\{$	`\biggl\{`	$\biggr\}$	`\biggr\}`	$\biggl($	`\biggl(`	$\biggr)$	`\biggr)`
$\Biggl\{$	`\Biggl\{`	$\Biggr\}$	`\Biggr\}`	$\Biggl($	`\Biggl(`	$\Biggr)$	`\Biggr)`

If you want, you can let TeX choose the size of delimiter by using the control words `\left` and `\right` before your delimiters. Thus `\left[...\right]` will cause the material TeXbook: 148 to be enclosed by brackets that are appropriately big. **Note well:** each use of a `\left` delimiter must have a matching `\right` delimiter (although the delimiters themselves may be different). Hence `$$\left|{a+b \over c+d}\right|.$$` gives

$$\left|{a+b \over c+d}\right|.$$

Mathematical delimiters

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

▹ Exercise 5.23 Typeset $\lceil\lfloor x\rfloor\rceil \le \lfloor\lceil x\rceil\rfloor$.

5.7 Those special functions

There are several types of functions that appear frequently in mathematical text. In an equation like "$\sin^2 x + \cos^2 x = 1$" the trigonometric functions "sin" and "cos" are in roman rather than italic type. This is the usual mathematical convention to indicate that it is a function being described and not the product of three variables. The control words `\sin` and `\cos` will use the right typeface automatically. Here is a table of these and some other special functions: TEXbook: 162

Special mathematical functions

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

▹ Exercise 5.24 Typeset: $\sin(2\theta) = 2\sin\theta\cos\theta \quad \cos(2\theta) = 2\cos^2\theta - 1$.

▷ Exercise 5.25 Typeset:

$$\int \csc^2 x\,dx = -\cot x + C \qquad \lim_{\alpha\to 0} \frac{\sin\alpha}{\alpha} = 1 \qquad \lim_{\alpha\to\infty} \frac{\sin\alpha}{\alpha} = 0.$$

▷ Exercise 5.26 Typeset:

$$\tan(2\theta) = \frac{2\tan\theta}{1-\tan^2\theta}.$$

5.8 Hear ye, hear ye!

There is a particular macro that is used in almost every mathematical paper, and is different enough to require a special explanation. This is the `\proclaim` macro. It is used when stating theorems, corollaries, propositions, and the like. The paragraph following `\proclaim` is broken into two parts: the first part goes up to and including the first period that is followed by a space, and the second part is the rest of the paragraph. The idea is that the first part should be something like "Theorem 1." or "Corollary B." The second part is the statement of the theorem or corollary. Here is an example:

TEXbook: 202–203

```
\proclaim Theorem 1 (H.~G.~Wells).  In the country of the blind,
the one-eyed man is king.
```

gives

Theorem 1 (H. G. Wells). *In the country of the blind, the one-eyed man is king.*

The statement of the theorem may contain mathematical expressions, of course.

▷ Exercise 5.27 Typeset:

Theorem (Euclid). *There exist an infinite number of primes.*

▷ Exercise 5.28 Typeset:

Proposition 1. $\sqrt[n]{\prod_{i=1}^n X_i} \le \frac{1}{n}\sum_{i=1}^n X_i$ *with equality if and only if* $X_1 = \cdots = X_n$.

5.9 Matrices

Matrices are typeset using combinations of the alignment character `&` and the control word `\cr` to indicate the end of the line. Start with `$$\pmatrix{...}$$`. Into the space between the braces go the rows of the matrix, each one ended by `\cr`. The entries are separated by the `&`. For example the input

TEXbook: 176–178

```
$$\pmatrix{
a & b & c & d \cr
b & a & c+d & c-d \cr
0 & 0 & a+b & a-b \cr
0 & 0 & ab & cd \cr
}.$$
```

gives as printed output

$$\begin{pmatrix} a & b & c & d \\ b & a & c+d & c-d \\ 0 & 0 & a+b & a-b \\ 0 & 0 & ab & cd \end{pmatrix}.$$

The matrix entries in our examples have all been centred within their columns with a little space on each side. They can be made flush right or flush left by inserting `\hfill` before or after the entry. Notice the differences between the following example and the previous one.

```
$$\pmatrix{
a & b & c \hfill & \hfill d \cr
b & a & c+d & c-d \cr
0 & 0 & a+b & a-b \cr
```

```
0 & 0 & ab \hfill & \hfill cd \cr
}.$$
```

gives as printed output

$$\begin{pmatrix} a & b & c & \hfill d \\ b & a & c+d & c-d \\ 0 & 0 & a+b & a-b \\ 0 & 0 & ab \hfill & \hfill cd \end{pmatrix}.$$

▷ Exercise 5.29 Typeset

$$I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

It's possible to have matrices that use other delimiters. Using `\matrix` instead of `\pmatrix` will leave off the parentheses, so the delimiters must be explicitly included using `\left` and `\right`. Here is how we can change the matrix of our first example.

```
$$ \left |
\matrix{
a & b & c & d \cr
b & a & c+d & c-d \cr
0 & 0 & a+b & a-b \cr
0 & 0 & ab & cd \cr
}
\right | $$
```

gives as printed output

$$\left| \matrix{ a & b & c & d \cr b & a & c+d & c-d \cr 0 & 0 & a+b & a-b \cr 0 & 0 & ab & cd \cr } \right|$$

It's even possible to use `\left.` and `\right.` to indicate that the opening or closing delimiter is deleted (note the use of the period).

▷ Exercise 5.30 Use a matrix construction to typeset

$$|x| = \begin{cases} x & x \geq 0 \\ -x & x \leq 0 \end{cases}$$

This exercise and more general constructions of this type may also be typeset using the `\cases` macro. TEXbook: 175

Sometimes ellipses are used within matrices. The control words `\cdots`, `\vdots`, and `\ddots` can be used to insert horizontal, vertical, and diagonal dots.

Thus we can use

```
$$ \left [
\matrix{
aa & \cdots & az \cr
\vdots & \ddots & \vdots \cr
za & \cdots & zz \cr
}
\right ] $$
```

to get as printed output

$$\left[\matrix{aa & \cdots & az \cr \vdots & \ddots & \vdots \cr za & \cdots & zz \cr}\right]$$

Matrices may also be typeset in-line, but they are pretty ugly unless they have a small number of rows.

5.10 Displayed equations

All of the mathematics covered so far has identical input whether it is to be typeset in-line or displayed. At this point we'll look at some situations that apply to displayed equations only.

The first is that of aligning multiline displays. This is done with the alignment character `&` and the control words `\cr` and `\eqalign`. Starting with `$$\eqalign{...}$$`, the equations to be aligned are entered with each one terminated by `\cr`. In each equation there should be one alignment symbol & to indicate where the alignment should take place. This is usually done at the equal signs, although it is not necessary to do so. For example TEXbook: 190–192

```
$$\eqalign{
a+b &= c+d \cr
x &= w + y + z \cr
m + n + o + p &= q \cr
}$$
```

yields

$$\begin{aligned} a+b &= c+d \\ x &= w+y+z \\ m+n+o+p &= q \end{aligned}$$

Displayed equations can be numbered at either the right or left margin. When the control word `\eqno` appears in a displayed equation, everything after the control word is put at the right margin. Hence `$$ x+y=z.  \eqno (1)$$` yields

$$x+y=z. \tag{1}$$

To number an equation at the left margin, use `\leqno` in place of `\eqno`.

It's possible to number aligned equations by using the control word `\eqalignno`. The alignment character & is used to separate the equation from the equation number.

```
$$\eqalignno{
a+b &= c+d & (1) \cr
x &= w + y + z \cr
m + n + o + p &= q & * \cr
}$$
```

yields

$$\begin{aligned} a+b &= c+d \qquad (1) \\ x &= w+y+z \\ m+n+o+p &= q \qquad * \end{aligned}$$

Use `\leqalignno` to put the equation numbers on the left. TEXbook: 192–193

Finally, suppose some text needs to appear in the middle of a displayed equation. This can be done by putting it in an hbox. We will describe hboxes in more detail in the next section. For now we want to use them to temporarily resume using ordinary roman type and to also allow the insertion of space between words (remember that all spaces are ignored when typesetting mathematics). Hence `$$X=Y \hbox{ if and only if }x=y.$$` will give

$$X = Y \text{ if and only if } x = y.$$

Note carefully the spaces in the hbox.

▷ Exercise 5.31 Do some of the challenge problems on pages 180–181 of The TEXbook.

Section 6
All in a row

It's not uncommon to want to put a table in the middle of some text. Fortunately TEX makes it easy to do this. In fact there are two separate methods of aligning text. The first is by using the tabbing environment. This is similar to setting the tab stops on a typewriter. Each line is handled individually, according to set tab columns, but with greater flexibility than that provided by a typewriter. The second is the horizontal alignment environment which typesets the whole table at once using a prescribed pattern.

6.1 Picking up the tab

To align material using the tabbing environment, you must first set the tab positions using the `\settabs` control word. Having done this, a line to use these tabs starts with the control symbol `\+` and ends with `\cr` (remember that the actual spacing on lines in the input file in unimportant).

The easiest way to use the `\settabs` control word is to put the text into equal columns. Using `\settabs 4 \columns` will set the tabs that will produce four equal columns. The tabbing is then done by using the alignment character `&` to move to the next tab stop. So, for example, TEXbook: 231

```
\settabs 4 \columns
\+ British Columbia & Alberta & Saskatchewan & Manitoba \cr
\+ Ontario & Quebec & New Brunswick & Nova Scotia \cr
\+ & Prince Edward Island & Newfoundland \cr
```

will produce the table

British Columbia	Alberta	Saskatchewan	Manitoba
Ontario	Quebec	New Brunswick	Nova Scotia
	Prince Edward Island	Newfoundland	

Notice that it is possible to skip over some tab positions, and it is not necessary to use all of the tabs in a given line. To make the same table using five columns, it is only necessary to use `\settabs 5 \columns` to reset the tab stops; then the same three lines from the last example will produce:

British Columbia	Alberta	Saskatchewan	Manitoba
Ontario	Quebec	New Brunswick	Nova Scotia
	Prince Edward Island	Newfoundland	

In this example, the columns are smaller, of course. In fact there are two overlapping entries in the last row. This is because TEX will tab to the next tab position even if (unlike a typewriter) it means going backward on the page.

There is an interesting relationship between grouping and tabbing. The `\settabs` values are only applicable to the group in which it is defined, as would be expected. Thus it is possible to temporarily change the tab settings by grouping within braces. In addition, each table entry is in a group of its own. Hence we may make a single entry boldface, for example, by using `\bf` without braces. In addition, for any column but the last one it is possible to centre the entry or to align it either on the left or on the right, or to fill a column with a line or dots. Each entry has an implicit `\hfil` at the end so that it will be at the left of the column by default. Adding `\hfil` at the beginning of the entry will then cause it to be centred, just as with the `\line` control word. Adding `\hfill` to the beginning will cause the entries to be pushed to the right (`\hfill` acts just like `\hfil` in that it absorbs excess space; when both `\hfil` and `\hfill` appear, the `\hfill` takes precedence).

```
\settabs 4 \columns
\+ \hfil British Columbia & \hfill Alberta \qquad & \bf Saskatchewan
                                  & Manitoba \cr
\+ \hfil Ontario & \hfill Quebec \qquad & \bf New Brunswick
                                  & Nova Scotia \cr
\+ \hfil --- & \hfill * \qquad & \bf Newfoundland
                                  & Prince Edward Island \cr
\+ \dotfill && \hrulefill & \cr
```

will produce a table with the first column centred, the second column flush right with a `\qquad` of padding, and the third column boldface. The control words `\dotfill` and `\hrulefill` give alternative column entries.

British Columbia	Alberta	**Saskatchewan**	Manitoba
Ontario	Quebec	**New Brunswick**	Nova Scotia
—	*	**Newfoundland**	Prince Edward Island
....			

▹ Exercise 6.1 Take the table of Canadian provinces above and centre each entry within its column.

The tab positions can be set with much more flexibility than just in equal columns. The general pattern is to use a sample line of the form `\settabs \+ ... & ... & ... \cr`. The spacing between the alignment characters `&` determines the position of the tabs. For example, `\settabs \+ \hskip 1 in & \hskip 2 in & \hskip 1.5 in & \cr` would set the first tab one inch from the left margin, the next another two inches further in, and the third 1.5 inches more. It's also possible to use text to determine the distance between tabs. So, for example, another possible sample line is `\settabs \+ \quad Province \quad & \quad Population \quad & \quad Area \quad & \cr`. The tab column would then be just wide enough to accept the headings with a quad of space on each side. Here's a more complete example:

```
\settabs \+ \quad Year \quad & \quad Price \quad
& \quad Dividend & \cr
\+ \hfill Year \quad & \quad Price  \quad & \quad Dividend \cr
\+ \hfill 1971 \quad & \quad 41--54 \quad & \qquad \$2.60 \cr
\+ \hfill 2    \quad & \quad 41--54 \quad & \qquad \$2.70 \cr
\+ \hfill 3    \quad & \quad 46--55 \quad & \qquad \$2.87 \cr
\+ \hfill 4    \quad & \quad 40--53 \quad & \qquad \$3.24 \cr
\+ \hfill 5    \quad & \quad 45--52 \quad & \qquad \$3.40 \cr
```

gives TEXbook: 247

Year	Price	Dividend
1971	41–54	\$2.60
2	41–54	\$2.70
3	46–55	\$2.87
4	40–53	\$3.24
5	45–52	\$3.40

▹ Exercise 6.2 Take the table given above and move it closer to the centre of the page.

▹ Exercise 6.3 One way to centre a block of text, possibly several lines long, is to use: `$$\vbox{...}$$`. Use this to centre the table given above. Does the `\settabs` line need to be included in the `\vbox`?

▹ Exercise 6.4 Improve your last result by putting a line under the column heads. The control word `\hrule` will insert a horizontal line if introduced between two rows of a table. Now repeat with the control word `\strut` after the `\+` of the line containing the column heads. (A `\strut` effectively makes the spacing between lines a little greater. The size can be altered from the default.) Note the extra space that results. TEXbook: 82

▷ Exercise 6.5 Make the following table with decimal alignment, that is, with the decimal points above each other (think of the dollar figure as being right aligned and the cents figure as being left aligned against the decimal point):

Plums	$1.22
Coffee	1.78
Granola	1.98
Mushrooms	.63
Kiwi fruit	.39
Orange juice	1.09
Tuna	1.29
Zucchini	.64
Grapes	1.69
Smoked beef	.75
Broccoli	1.09
Total	$12.55

▷ Exercise 6.6 Devise a method to make a rough table of contents by using `\settabs` and having entries looking something like:

```
Getting Started \dotfill & \hfill 1
All Characters Great and Small \dotfill & \hfill 9.
```

6.2 Horizontal alignment with more sophisticated patterns

The `\settabs` environment is not difficult to use, and once the pattern is set, it can be used repeatedly in different portions of the text that follows. It does have some drawbacks, however. For one, the column size must be set before the entries are known. Also, even though in one case we wanted the third column to be boldface, it had to be specified in each line. These problems can be handled more easily by using the `\halign` environment. TEXbook: 235–238

The general pattern in the `\halign` is as follows:

```
\halign{ <template line> \cr
<first display line> \cr
<second display line> \cr
```

```
⋮
<last display line> \cr
}
```

Both the template line and the display lines are divided into sections by the alignment symbol `&`. In the template line each section uses control words in the same manner as does `\line{}`. The control word `\hfil`, for example, can be used to display flush left, flush right, or centred. Fonts can be changed using `\bf`, `\it`, etc. Text may also be entered in the template line. In addition the special symbol `#` must appear once in each section. Each display line is then set by substituting each section of the display line into its corresponding section of the template line at the occurrence of the `#`.

Consider the following example:

```
\halign{\hskip 2 in $##$& \hfil \quad # \hfil & \qquad $##$
                                        & \hfil \quad # \hfil \cr
\alpha & alpha & \beta & beta \cr
\gamma & gamma & \delta & delta \cr
\epsilon & epsilon & \zeta & zeta \cr
}
```

The template line indicates that the first section of the typeset text will always be set two inches in from the left and also be set as mathematics. The second section will be centred after adding a quad of space on the left. The third and fourth sections are handled similarly. Here is the result:

α	alpha	β	beta
γ	gamma	δ	delta
ϵ	epsilon	ζ	zeta

In this case the first display line is formed by substituting `\alpha` for the first `#` in the template line, `alpha` for the second `#`, `\beta` for the third and `beta` for the fourth. The whole line is then saved for setting. This continues until all the lines are accumulated, and then they are set with each column being as wide as necessary to accept all of its entries (an implication of this accumulation process is that a table with too many entries could cause TeX to run out of memory; it's better not to set tables that are more than a page or so long).

Hence the template line establishes the pattern for the table entries and the display lines insert the individual entries.

Sometimes horizontal and vertical lines are used to delimit entries in a table. To put in horizontal lines, we use `\hrule`, just as we did in the `\settabs` environment. However, we don't want the rule to be aligned according to the template, so we use the control word `\noalign`. Hence horizontal lines are inserted by putting `\noalign{\hrule}`; vertical lines are inserted by putting `\vrule` in either the template or the display line. But still all is not completely straightforward. Suppose we take our last example and change the template to get vertical lines and also insert horizontal lines.

```
\halign{\hskip 2in\vrule\quad $##$\quad & \vrule \hfil\quad # \hfil
                           & \quad \vrule \quad $##$\quad
                           & \vrule\hfil \quad # \quad \hfil \vrule \cr
\noalign{\hrule}
\alpha & alpha & \beta & beta \cr
\noalign{\hrule}
\gamma & gamma & \delta & delta \cr
\noalign{\hrule}
\epsilon & epsilon & \zeta & zeta \cr
\noalign{\hrule}
}
```

doesn't give exactly what we want.

	α	alpha	β	beta
	γ	gamma	δ	delta
	ϵ	epsilon	ζ	zeta

There are several deficiencies: the most obvious is the extended horizontal lines, but also the text looks somewhat squashed into the boxes. In addition, the text has a little extra space on the right rather than being perfectly centred. As in the `\settabs` environment, lines can be made taller by including the control word `\strut` in the template. A further problem can occur when the page is set since TEX may spread lines apart slightly to improve the appearance of the page. This would leave a gap between the vertical lines, so we use the control word `\offinterlineskip` within the `\halign` to avoid this. Finally we can get rid of the lines sticking out on the left by deleting the `\hskip 2 in` from the template line. To move the table to the same position we use `\moveright`. Finally, we can see how to centre the text by noting that the extra space occurs in the template line after the `#` where the text is inserted. Hence we can improve our result by using

TEXbook: 82

```
\moveright 2 in
\vbox{\offinterlineskip
\halign{\strut \vrule \quad $##$\quad &\vrule \hfil \quad #\quad \hfil
&\vrule \quad $##$\quad &\vrule \hfil \quad #\quad \hfil \vrule \cr
\noalign{\hrule}
```

```
\alpha & alpha & \beta & beta \cr
\noalign{\hrule}
\gamma & gamma & \delta & delta \cr
\noalign{\hrule}
\epsilon & epsilon & \zeta & zeta \cr
\noalign{\hrule}
}}
```

to get

α	alpha	β	beta
γ	gamma	δ	delta
ϵ	epsilon	ζ	zeta

In general, if we want to construct a table with boxed entries that is centred on the page, we can do so by putting the `\vbox` within a `\centerline{}`. But here is a trick that will produce a nicer result. If the `\vbox` is put in between double dollar signs, it will be typeset as displayed mathematics. Of course, there is no actual mathematics being displayed, but TEX will put in a little extra space above and below the table as is appropriate for a display. Hence a centred table with this nice spacing may be formed using the following four steps: (1) put a `\vbox` between double dollar signs, (2) put an `\offinterlineskip` and an `\halign` within the `\vbox`, (3) in the `\halign` put a template line with a `\strut` in the beginning, and a `\vrule` surrounding each entry, (4) each row of the table should be preceded and followed by `\noalign{\hrule}`.

Here is the pattern to be followed:

```
$$\vbox{
\offinterlineskip
\halign{
\strut \vrule # & \vrule # & ...& \vrule # \vrule \cr
\noalign{\hrule}
<first column entry> & <second column entry> & ...& <last column entry> \cr
\noalign{\hrule}
...
\noalign{\hrule}
<first column entry> & <second column entry> & ...& <last column entry> \cr
\noalign{\hrule}
}
}$$
```

Section 7
Rolling your own

In this section we'll create new control words. The making of these new definitions, also called macros, is one of the most powerful techniques available in TeX. For the first application of this facility, we'll see how a new definition can save a lot of typing by substituting short strings for long ones.

7.1 The long and short of it

The control word `\def` is used to define new control words. The simplest form for doing this is `\def\newname{...}`. Then whenever `\newname` appears in your input file, it will be replaced by whatever is between the braces in the definition. Of course `\newname` must satisfy the convention for naming control sequences, that is, it must be a control word (all letters) or a control symbol (exactly one nonletter). So, for example, suppose you write a document that contains the phrase "University of Manitoba" many times. Then `\def\um{University of Manitoba}` defines a new control sequence `\um` which can then be used at any time. The sentence `I take courses at the \um.` then makes sense. If the control word exists, your new definition will replace it (this includes the control words defined by TeX, so a little care must be taken in the choice of name). Any definition is, however, local to the group in which it is defined. For example,

```
\def\um{University of Manitoba}
I took my first course at the \um.
{
\def\um{Universit\'e de Montr\'eal}
Then I took my next course at the \um.
}
Finally I took my last course at the \um.
```

gives

I took my first course at the University of Manitoba. Then I took my next course at the Université de Montréal. Finally I took my last course at the University of Manitoba.

Remember that all spaces after a control word are absorbed; this includes the control words that you define. In the previous example, any space after `\um` would be ignored. However, the space after the first period and the space after the first opening brace are different; if you look closely at the end of the first sentence typeset using the example, you'll see some extra space. This can be eliminated by putting a `%` after the opening brace to make the rest of the line a comment. The same holds for the line with the last closing brace. Careful control of spaces often calls for the "commenting out" of the end of lines in this manner.

Once a new control sequence has been defined, it may be used in new definitions. This is one way of making simple form letters. First let's define a simple letter.

```
\def\letter{
\par \noindent
Dear \name,
 This is a little note to let you know that your name is \name.
 \hskip 2 in Sincerely yours,
\vskip 2\baselineskip
\hskip 2 in The NameNoter
\smallskip \hrule
}
```

Now this letter uses the control word `\name`, which is undefined at this point. When `\letter` is used, the current value of `\name` will appear in the body of the letter. Hence

```
\def\name{Michael Bishop}
\letter
\def\name{Michelle L\'ev\^eque}
\letter
```

will produce two copies of the letter, each with the correct name, followed by a horizontal rule:

Dear Michael Bishop,

This is a little note to let you know that your name is Michael Bishop.

Sincerely yours,

The NameNoter

Dear Michelle Lévêque,

This is a little note to let you know that your name is Michelle Lévêque.

Sincerely yours,

The NameNoter

We could have put anything between the braces in `\def\name{...}`; it could be several paragraphs long and use other control sequences (although in this context it would be a little strange). Of course it is possible to use `\vfill \eject` as part of the definition of `\letter` to eject the page when the letter is completed.

▷ Exercise 7.1 Make a form letter that uses the control words `\name`, `\address`, `\city`, `\province`, and `\postalcode`.

▷ Exercise 7.2 An unnumbered list of items is often made using `\item{$\bullet$}`. Define a macro `\bitem` that does this, and use it for several paragraphs. Now change each bullet to a dash (note that a simple change in the macro propagates all the necessary changes in all of the paragraphs).

▷ Exercise 7.3 Suppose that you are going to have to format several paragraphs in a paper using `\hangindent = 30 pt`, `\hangafter = 4`, and `\filbreak` (don't worry about what these control sequences actually do; the only important thing for now is that once they are set, they remain in effect for only one paragraph). Define a single control sequence `\setpar` which can then be put in front of each paragraph that needs to be so formatted.

7.2 Filling in with parameters

It's possible to use macros in much greater generality by allowing parameters to be passed. The idea is somewhat similar to the template line in the `\halign` environment. First, let's look at the case where there is one parameter. In this case a control sequence is defined by `\def\newword#1{...}`. The symbol `#1` may appear between the braces (several times) in the definition of `\newword`. The material between the braces acts like a template. When `\newword{...}` appears in the text, it will use the definition of `\newword` with the material between the braces inserted into the template at every occurrence of `#1` in the original definition. **The spacing in the original definition is crucial here; there must be no spaces before the opening brace**.

As an example, we could use the form letter of the last section in the following way:

```
\def\letter#1{
\par \noindent
Dear #1,

This is a little note to let you know that your name is #1.

\hskip 2 in Sincerely yours,
\vskip 2\baselineskip
\hskip 2 in The NameNoter
\smallskip \hrule
}
```

Now we can use

```
\letter{Michael Bishop}
\letter{Michelle L\'ev\^eque}
```

to get

Dear Michael Bishop,

This is a little note to let you know that your name is Michael Bishop.

Sincerely yours,

The NameNoter

Dear Michelle Lévêque,

This is a little note to let you know that your name is Michelle Lévêque.

Sincerely yours,

The NameNoter

Now let's define `\def\displaytext#1{$$\vbox{\hsize = 12 cm #1}$$}` as a new macro to display text. Then `\displaytext{...}` will cause the material between the braces to be put in a paragraph with width 12 centimetres and then centred with some space added above and below as is appropriate for a display. This paragraph was set using this `\displaytext` macro.

The parameter of a macro can be no more than one paragraph long. If a new paragraph is encountered as part of a parameter, an error will be generated. This is a safety feature, for otherwise the accidental omission of a closing brace would cause TEX to eat up the rest of the file as the parameter.

▹ Exercise 7.4 Define a macro `\yourgrade` so that `\yourgrade{89}` will cause the following sentence to be typeset: The grade you received is 89%. It should be able to work with any other percentage, of course.

It's not really any harder to use more than one parameter. The form used to define a new control word with two parameters is `\def\newword#1#2{...}`. The definition between the braces may have `#1` and `#2` occurring in it, perhaps several times. When `\newword{...}{...}` appears in the text, the material between the first set of braces replaces `#1` in the definition and the material between the second set of braces replaces `#2` in the definition. Here is an example followed by its result:

```
\def\talks#1#2{#1 talks to #2.}
\talks{John}{Jane}
\talks{Jane}{John}
\talks{John}{me}
\talks{She}{Jane}
```

John talks to Jane. Jane talks to John. John talks to me. She talks to Jane.

▷ Exercise 7.5 In a manner similar to the previous exercise, define a macro `\yourgrade` so that `\yourgrade{89}{85}` causes the following sentence to be typeset: You received a grade of 89% on your first exam and a grade of 85% on your second exam.

▷ Exercise 7.6 Write a macro `\frac` so that `\frac{a}{b}` will typeset the fraction $\frac{a}{b}$.

It's important not to put any spaces before the first brace in the definition. If you do, TEX will interpret the definition differently from the way described here. For more than two parameters, the method of definition is similar. To define a control word with three parameters, start with `\def\newword#1#2#3{...}`. Then `#1`, `#2` and `#3` may occur between the braces. When `\newword{...}{...}{...}` appears in the text, the material between each set of braces replaces its corresponding symbol in the definition of the control word. The parameters may go up to `#9`.

7.3 By any other name

Sometimes it's convenient to be able to give a control word an alternative name. For example, if you prefer a different spelling, you might want to call the control word `\centerline` by the name of `\centreline`. This can be done by using the `\let` control word. The use of `\let \centreline = \centerline` now makes a new (as well as the old) control word available. This can also be used with mathematical names as with `\let \tensor = \otimes`. It is then possible to use TEXbook: 206–207

```
$$ (A \tensor B) (C \tensor D) = AC \tensor BD. $$
```

to get

$$(A \otimes B)(C \otimes D) = AC \otimes BD.$$

▷ Exercise 7.7 Define control sequences `\ll`, `\cl`, and `\rl` that are equivalent to `\leftline`, `\centerline`, and `\rightline`.

The `\let` control word allows users to name their own control sequences. This allows a personalized set of control sequences that may be used in place of the ones provided by TeX when desired.

Section 8
To err is human

In some ways TeX is not completely divine. TeX will respond to invalid input by giving an error message to the screen if you are using it interactively and also to the log file. Because TeX is very complicated, the actual point where the error is detected may be deep within the program, so a full report of the error may be rather long and involved. Not only that, TeX will try to recover from errors, and will report what was done in that process. For this reason the reading of error messages may be a little difficult for the uninitiated. The key is to know what is important from your perspective and what can be safely ignored. So let's look at some typical errors and the messages that they generate.

8.1 The forgotten bye

The first mistake that we'll look at is one that everyone makes at some time, namely, the omission of `\bye` at the end of the file. If you're using TeX interactively, an asterisk

```
*
```

will be printed on the screen and nothing will happen since, having not been told to finish up, TeX is waiting for input (from your keyboard). Whatever you type in will be appended to whatever has been input from your files. The usual response is to type `\bye<CR>`[7] since that will finish things up.

8.2 The misspelled or unknown control sequence

Using a misspelled or other control sequence unknown to TeX is a common error. If TeX is being run as a batch job, an error message is printed and the job goes on ignoring the control sequence. When using TeX interactively, it is possible to repair errors (of course this does not change the original input file, so that must be done when the TeX job is completed). Suppose we have a TeX input file consisting of the following two lines:

```
\line{The left side \hfli the right side}
\bye
```

[7] `<CR>` is the key used to end a line of input. It might be called the carriage return, enter, or simply the return key on your terminal. Sometimes it is indicated by a large left arrow.

The control word should be `\hfil`, of course. Here is the message that would be sent to your terminal:

```
! Undefined control sequence.
l.1 \line{ The left side \hfli
                               the right side}
?
```

The first line starts with `!` and gives the error message. Next comes the line number on which the error occurred and the part of the line that was read successfully. The next line gives the continuation of the line after the error. At this point the question mark means that TEX is waiting for a response. There are several legal ones:

Responses to TEX error messages

Desired response	Input to TEX	Result
Help	`h<CR>`	Reason for stopping listed on terminal.
Insert	`i<CR>`	Next line inserted into TEX input file.
Exit	`x<CR>`	Exit from TEX. Completed pages to DVI file.
Scroll	`s<CR>`	List message and continue after minor errors.
Run	`r<CR>`	List message and continue after any errors.
Quiet	`q<CR>`	All terminal listings suppressed.
Carry on	`<CR>`	TEX continues as best it can.

In our last example a reasonable response might be to enter `h<CR>` to get a help message, then `i<CR>` to insert more text, (at which point TEX responds with `insert>` and finally `\hfil` as the correct control word. Here is the result:

```
? h <CR>
The control sequence at the end of the top line
of your error message was never \def'ed. If you have
misspelled it (e.g., '\hobx'), type 'I' and the correct
spelling (e.g., 'I\hbox'). Otherwise just continue,
and I'll forget about whatever was undefined.
? i <CR>
insert>\hfil
[1]
```

The final `[1]` means that the first (and only) page has been completed and sent to the DVI file. The original input file still needs to be fixed, of course.

8.3 The misnamed font

A misspelled font name is an error similar to the misspelled control sequence. The error message is different and a little confusing at first. Suppose for example the following appears in your input file:

```
\font\sf = cmss01
```

It should be `cmss10`, that is, the numbers have been transposed. Here are the error and help messages:

```
!  Font \sf=cmss01 not loadable:  Metric (TFM) file not found.
<to be read again>
                   \par
\bye ->\par
            \vfill \supereject \end
l.2 \bye
?  h <CR>
I wasn't able to read the size data for this font,
so I will ignore the font specification.
[Wizards can fix TFM files using TFtoPL/PLtoTF.]
You might try inserting a different font spec;
e.g., type 'I\font<same font id>=<substitute font name>'.
```

The TFM (TEX font metric) file is an auxiliary file that is used by TEX. So this strange message is just telling you that the font you defined doesn't exist on your computer system.

8.4 Mismatched mathematics

Another common error is to use `$` or `$$` to start a mathematical expression and then to forget the second `$` or `$$` when finished. The text that follows is then treated as mathematics, and to make matters worse, if more mathematics is started by a new `$` or `$$`, it will then be treated as ordinary text. Needless to say, error messages galore may be generated. TEX will attempt to correct the problem by inserting a new `$` or `$$`; in any case, the problem is corrected by the end of the paragraph since a new paragraph will automatically start as ordinary text.

Consider the following correct input and its output:

```
Since $f(x) > 0$, $a<b$, and $f(x)$ is continuous, we know that
$\int_a^b f(x)\,dx >0$.
```

Since $f(x) > 0$, $a < b$, and $f(x)$ is continuous, we know that $\int_a^b f(x)\,dx > 0$.

If we now leave out the second dollar sign in `$f(x)$` we then get the following error and help messages:

```
!  Missing $ inserted.
<inserted text>
               $
<to be read again>
                  \intop
\int ->\intop
             \nolimits
l.2 $\int
          _a^b f(x)\,dx >0$.
?  h <CR>
I've inserted a begin-math/end-math symbol since I think
you left one out.  Proceed, with fingers crossed.
?
```

The line starting with `!` tells us what has been done. The line starting with `l.2` shows us where we were in the input file when the error occurred. As in our other examples, the part of the line successfully read, that is, through `\int`, appears on one line, and the continuation appears on the next line. The remaining material may seem somewhat obscure. These intermediate messages show what was happening further in the guts of the TEX program when the error occurred. The newer user may ignore them.

Here is what you get as output after TEX tries to recover from the error.

Since $f(x) > 0$, $a < b$, and $f(x) is continuous, we know that \int_a^b f(x)\,dx > 0$.

There is a stretch of text that is italic with no spacing. This is typical for normal text being processed as mathematics; if you see this in your output, you have almost certainly left out a `$` or `$$`.

8.5 Mismatched braces

It's easy to forget or mismatch the closing braces when making groups. The result may be a relatively benign error, or it may be catastrophic. Suppose, for example, you have `{\bf A bold title` in your text with the closing right brace omitted. The result will be the same as if no opening brace were there; that is, the rest of the paper will be boldface if no other font changes are made. You will get the following message at the end of the file:

```
(\end occurred inside a group at level 1)
```

If you had made the same mistake twice, then there would be two more opening braces than closing braces, and you would get the message:

```
(\end occurred inside a group at level 2)
```

TEX doesn't know that the closing brace is missing until it reaches the end of the input file. Hence the message doesn't tell you where you went wrong. If the location of the missing brace isn't obvious, it's always possible to insert `\bye` halfway through your document. Running TEX again will cause only the first half to be processed, and if the error message persists, you will know that the error is in the first half of the document. By moving the `\bye` to different places, the error can be localized. Also, looking at the output often reveals what has gone wrong.

Missing opening braces are much easier to spot. Here is a two line input file and the resulting error and help messages:

```
\bf Here is the start}, but there is the finish.
\bye
```

```
!  Too many }'s.
l.1 \bf Here is the start}
                          , but there is the finish.
?  h <CR>
You've closed more groups than you opened.
Such booboos are generally harmless, so keep going.
```

It's quite possible, of course, that the line that is supposed to have the missing left brace will not be on the line where TEX catches the error.

A mismatched brace in the definition of a new control sequence can cause a major error. Since such a definition may include several paragraphs, it may not be caught by the end of a paragraph, but, rather will just keep piling more and more text into the unfinished definition. It's even possible for TEX to run out of memory as it keeps eating up more text! This is called a "runaway definition". Here is a two line input file with a runaway TEXbook: 206
definition:

```
\def\newword{the def
\newword
\bye
```

Here are the resulting error and help messages:

```
Runaway definition?
->the def
!  Forbidden control sequence found while scanning definition of \newword.
<inserted text>
                }
<to be read again>
                   \bye
l.3 \bye
?  h <CR>
I suspect you have forgotten a `}', causing me
to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
you'd better type `E' or `X' now and fix your file.
?  <CR>
No pages of output.
```

This is obviously a serious error. If it occurs at the beginning of a file (as in the previous example), there will be no output at all!

If a closing brace is left out while using a macro with parameters, the runaway definition will be terminated at the end of the paragraph. So if `\def\newword#1{...}` has been defined and you use `\newword{...` without the closing brace, then at most one paragraph will be ruined. TEXbook: 205

In short, when an error occurs, make a note of the line number to see how much of the input file has been read, and also the line starting with an exclamation point to get a short description of the error. If the error is still not clear, ask TEX for more information by typing `h<CR>`. For small errors, TEX can carry on quite a way if you just keep hitting the `<CR>`.

Section 9
Digging a little deeper

In this section we look at a few topics that allow TEX to be used with greater flexibility or efficiency. As the documents being produced get longer, different techniques can help make their creation easier.

9.1 Big files, little files

TEX can read and write files as it runs. This makes it possible to use files that are smaller and more convenient to handle by creating a master file that reads the smaller files in the proper order. This document, for example, consists of ten sections plus an introduction. In addition, there are macros that are used for all sections. The macros can be put in a file called, say, `macros.tex`, the introduction can be put in `intro.tex`, and each section put in its own file. The control word `\input` is then used to read in a file. In general, `\input filename` will cause the file called `filename.tex` to be read in and processed immediately, just as if the text of `filename.tex` had been part of the file that read it in. This file may input other files. In fact it's often convenient to make a single file that reads in smaller pieces, perhaps as follows:

```
\input macros
\input intro
\input sec1
\input sec2
\input sec3
\input sec4
\input sec5
\input sec6
\input sec7
\input sec8
\input sec9
\input sec10
```

While the text is still being heavily edited, it's possible to process only some of the files by putting a `%` at the beginning of each line that contains a file to be skipped (this is sometimes called "commenting out" the unwanted files).

The `\input` control word also allows the use of predesigned macros. The macros for a memorandum, for example, might be put in a file called `memo.tex`. These macros might set up the right `\hsize`, `\vsize` and other parameters, and might stamp the time and date. Once this has been set up, all memoranda may be started with `\input memo` to make them come out with a common format.

Be sure that you don't have the control word `\bye` in your input file or the TEX program will stop at that point.

▹ Exercise 9.1 Make a TEX input file that reads in a second file. Try reading in the second file twice using the `\input` control word twice.

9.2 Larger macro packages

Designing macros that can be used with many types of documents is obviously useful. Most universities, for example, have specific and often complicated format requirements for theses. A collection of macros, that is, a macro package that meets all these specifications could be somewhat time consuming to design and could be quite long. It is possible to use the `\input` command to use such a macro package, just as it is possible to use it with your own macros. But TEX has a better facility for larger packages.

A macro package can be put in a special form that can be quickly read by TEX. This is called a *format file*, and the exact form is of technical interest only. The important thing is it allows TEX to be run with many new control sequences predefined. Certain commands called *primitives* are part of the definition of TEX.

What we have described in this manual is sometimes called *plain TEX*, and consists of the TEX primitives plus a set of macros in a format file (that is usually included in TEX automatically) called `plain.fmt`. For the curious, any control word can be viewed using `\show`. The command `\show\centerline` will display

```
> \centerline=macro:
#1->\line {\hss #1\hss }.
```

on the screen and in the log file. You can use `\show` with your own macros, too. If you end up using several macro packages, you can use the `\show` command to see if a particular macro is defined.

Many computer centres have the LATEX macro package. This package allows the user to create an index, a table of contents, and a bibliography automatically. It also has the ability to insert some elementary graphic figures such as circles, ovals, lines, and arrows. LATEX also uses special predefined files called *style files* to set up specific page parameters. Many different style files are available; some journals will accept papers on a magnetic medium for direct processing if they are prepared using LATEX and a designated style file. It is not difficult to shift from TEX to LATEX. A user's guide by the author of the macro package, Leslie Lamport, is available: **LATEX: A document preparation system**[8].

The American Mathematical Society uses the $\mathcal{AMS}$-TEX macro package for its journals. It is readily available from that Society, and papers may be submitted to their journals on a magnetic medium using $\mathcal{AMS}$-TEX. A manual by Michael Spivak, **The Joy of TEX**[9], is available from the American Mathematical Society.

Other macro packages exist, and undoubtedly more will be developed. They are usually of modest cost and can be very effective in some circumstances. The TEX Users Group announces the existence of new macro packages in its publications.

9.3 Horizontal and vertical lines

Making horizontal and vertical lines is easy using TEX. When typing in text, `\hrule` will cause the current paragraph to end, will draw a horizontal line whose width is the current value of `\hsize`, and then will continue on with a new paragraph. It's possible to specify the width of the hrule as, for example, with `\hrule width 5 cm`; also you can use `\vskip` or `\bigskip` to put some space above or below the hrule. Here is an example:

```
\parindent = 0 pt \parskip = 12 pt
Here is the text before the hrule.
\bigskip
\hrule width 3 in
And here is some text after the hrule.
```

that produces

[8] Addison-Wesley, Reading, Massachusetts, 1986, ISBN 0-201-15790-X.
[9] American Mathematical Society, 1986, ISBN 0-8218-2999-8

Here is the text before the hrule.

And here is some text after the hrule.

In fact this hrule not only has width of three inches, but also by default has a height (the amount by which the hrule extends above the baseline on which the type is being set) of 0.4 points and a depth (the amount by which the hrule extends below the baseline on which the type is being set) of 0 points. Each of these parameters can be individually set. Thus if we change the last example to say

`\hrule width 3 in height 2 pt depth 3 pt` we get

Here is the text before the hrule.

And here is some text after the hrule.

The three parameters `width`, `height`, and `depth` may be given in any order. TEXbook: 221–222

A vrule may be defined analogously to an hrule by specifying the `width`, `height`, and `depth` if desired. But, unlike the hrule, the vrule will not automatically start a new paragraph when it appears. By default the vrule will be 0.4 points wide, and will be as high as the line on which it is being set. Hence TEXbook: 221–222

```
Here is some text before the vrule
\vrule\
and this follows the vrule.
```

will give

Here is some text before the vrule | and this follows the vrule.

▷ Exercise 9.2 Make three horizontal lines that are 15 points apart, 3 inches in length, and one inch in from the left margin.

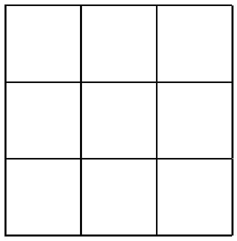
Although we usually think of hrules and vrules as horizontal and vertical lines, they need not necessarily be used that way. For example:

```
\noindent
Name:  \vrule height 0 pt depth 0.4 pt width 3 in
```

will give

Name: ______________________________

▷ Exercise 9.3 Make the following grid (each box is 1 centimetre square):



9.4 Boxes within boxes

We have already seen (in our discussion of line shapes) that vboxes and hboxes are objects that may be overfull or underfull. In this section we will look at these boxes in a bit more detail. They may be stacked or lined up to allow a variety of positions for text on the page.

An hbox is formed by using `\hbox{...}`. Once the material between the braces has been put into an hbox, it is set and can not be further split (this means that material that must go on one line can be put into an hbox, and it will then remain as one unit). It's possible to specify the size of an hbox. Thus `\hbox to 5 cm{contents of the box}` will produce an hbox exactly five centimetres wide containing the typeset text "contents of the box". It's easy to get an underfull or overfull box in this way. An underfull box can be avoided by using `\hfil` to absorb the extra space. When no dimension is given, an hbox is formed that is just wide enough to hold the enclosed text.

TEXbook: 64–66

Similarly, vboxes are formed using `\vbox{...}`. What makes these boxes interesting is that when a vbox contains hboxes, these hboxes are stacked one above the other and set as a unit. Similarly, an hbox can contain vboxes, which will be set in a row. Suppose we take three hboxes and put them in a vbox:

```
\vbox{
      \hbox{Contents of box 1}
      \hbox{Contents of box 2}
      \hbox{Contents of box 3}
```

```
      }
```

gives

Contents of box 1
Contents of box 2
Contents of box 3

Now suppose we take another vbox:

```
\vbox{
      \hbox{Contents of box 4}
      \hbox{Contents of box 5}
      }
```

These two vboxes can be put into an hbox; this will cause them to be placed side by side. In other words

```
\hbox{
     \vbox{
           \hbox{Contents of box 1}
           \hbox{Contents of box 2}
           \hbox{Contents of box 3}
           }
     \vbox{
           \hbox{Contents of box 4}
           \hbox{Contents of box 5}
           }
     }
```

gives

Contents of box 1
Contents of box 2 Contents of box 4
Contents of box 3 Contents of box 5

Notice that the two vboxes are aligned so that the bottoms are level; also there is a little space at the beginning of each line and also between the vboxes. Actually, the reason these spaces appear is rather subtle. Unless a line ends in a control word, there is always a space between the last entry in one line and the first one in the next line. For this reason the space between the vboxes comes from the end of the line containing the closing brace of the first vbox. Similarly, the space at the beginning of the line is caused by the space after the opening brace of the hbox. These spaces can be avoided by "commenting out" the end of the line, that is, by putting a `%` immediately after the closing brace of the first vbox or the opening brace of the hbox. If you try to put some vboxes together and accidentally get extra space by forgetting to comment out the end of the line, you're in good company. Some very able and experienced TEX users have done the same thing!

Extra space, say one centimetre, can be added by putting an `\hskip 1 cm` between the vboxes. They can be aligned so that the tops are level by using `\vtop` instead of `\vbox`. Making these two changes results in:

Contents of box 1	Contents of box 4
Contents of box 2	Contents of box 5
Contents of box 3	

We can combine vboxes, hboxes, vrules, and hrules to get boxed text. How might we construct such a box? One way is to take the material to be boxed and put it in an hbox preceded and followed by a vrule. Then put this in a vbox with hrules above and below it. This gives us:

```
\vbox{
      \hrule
      \hbox{\vrule{} The text to be boxed \vrule}
      \hrule
     }
```

which results in

$\boxed{\text{The text to be boxed}}$

This produces boxed material, but there is no margin around it and so it looks very cramped (of course TEX is just giving us what we asked for). We can improve the spacing by putting a `\strut` at the beginning of the hbox to make it a little taller and deeper. This gives us:

$\boxed{\text{The text to be boxed}}$

▷ Exercise 9.4 Why is it that we were forced to add extra space above and below the text but not before and after it?

▷ Exercise 9.5 Use the method of boxing material to put text centred in a box which extends from the left to the right margin.

▷ Exercise 9.6 By stacking nine little boxes, make the following magic square:

6	1	8
7	5	3
2	9	4

▷ Exercise 9.7 Notice that the magic square in the previous exercise has internal lines that are twice as thick as the outside ones. Also, there is a tiny space at the intersection of the internal lines. Fix up the magic square so this doesn't happen.

▷ Exercise 9.8 Write a macro `\boxtext#1{...}` which will take the text between the braces and put a box around it. Test your macro by making up a sentence with every other word boxed. I'm not quite sure why someone would do this since the result is pretty strange. Note how the baseline and the bottom of the surrounding boxes align.

It's easy to move boxes up, down, left, or right on the page. A `\vbox` can be moved to the right one inch by using `\moveright 1 in \vbox{...}`. To move it to the left, use `\moveleft`. Similarly, an `\hbox` can be moved up or down using `\raise` or `\lower`.

▷ Exercise 9.9 Rewrite the `\boxtext` macro from the previous exercise so that all of the text is aligned (hint: by default the depth of a strut is 3.5 points). This would give a sentence like the following: I'm not quite sure why someone would do this since the result is pretty strange.

It's possible to fill a box with either an hrule or with dots. The idea is to use `\hrulefill` or `\dotfill` in the hbox.

```
\hbox to 5 in{Getting Started\hrulefill 1}
\hbox to 5 in{All Characters Great and Small\hrulefill 9}
\hbox to 5 in{The Shape of Things to come\hrulefill 17}
\hbox to 5 in{No Math Anxiety Here!\hrulefill 30}
```

gives

Getting Started____1
All Characters Great and Small______________________________________9
The Shape of Things to come_______________________________________17
No Math Anxiety Here!____30

If `\hrulefill` is replaced by `\dotfill` we get

Getting Started 1
All Characters Great and Small 9
The Shape of Things to come 17
No Math Anxiety Here! 30

▷ Exercise 9.10 Make a boxed headline appear at the top of the page that is like the one used in this manual.

Section 10
Control word list

Here is a list of the control words given in this manual. If you want more detail about these words than is given here, check the index of **The TEX book**.

Control symbols

Control words

Section 11
I get by with a little help

Many of the exercises can be answered in several ways. If you like your way better than the way given below, by all means use it!

```
I like \TeX!
Once you get the hang of it, \TeX\ is really easy to use.
You just have to master the \TeX nical aspects.
```

I like TEX! Once you get the hang of it, TEX is really easy to use. You just have to master the TEXnical aspects.

```
Does \AE schylus understand \OE dipus?
```

Does Æschylus understand Œdipus?

```
The smallest internal unit of \TeX{} is about 53.63\AA.
```

The smallest internal unit of TEX is about 53.63Å.

```
They took some honey and plenty of money wrapped up in a {\it \$}5 note.
```

They took some honey and plenty of money wrapped up in a £5 note.

```
\'El\`eves, refusez vos le\c cons! Jetez vos cha\^\i nes!
```

Élèves, refusez vos leçons! Jetez vos chaînes!

```
Za\v sto tako polako pijete \v caj?
```

Zašto tako polako pijete čaj?

```
Mein Tee ist hei\ss.
```

Mein Tee ist heiß.

```
Peut-\^etre qu'il pr\'ef\`ere le caf\'e glac\'e.
```

Peut-être qu'il préfère le café glacé.

```
?`Por qu\'e no bebes vino blanco? !`Porque est\'a avinagrado!
```

¿Por qué no bebes vino blanco? ¡Porque está avinagrado!

```
M\'\i\'\j n idee\"en worden niet be\"\i nvloed.
```

Míj́n ideeën worden niet beïnvloed.

```
Can you take a ferry from \"Oland to \AA land?
```

Can you take a ferry from Öland to Åland?

```
T\"urk\c ce konu\c san ye\u genler nasillar?
```

Türkçe konuşan yeğenler nasillar?

```
I entered the room and---horrors---I saw both my father-in-law and my
mother-in-law.
```

I entered the room and—horrors—I saw both my father-in-law and my mother-in-law.

```
The winter of 1484--1485 was one of discontent.
```

The winter of 1484–1485 was one of discontent.

```
His ``thoughtfulness'' was impressive.
```

His “thoughtfulness” was impressive.

```
Frank wondered, ``Is this a girl that can't say `No!'?''
```

Frank wondered, “Is this a girl that can’t say ‘No!’?”

```
He thought, ``\dots and this goes on forever, perhaps to the last recorded
syllable.''
```

He thought, “. . . and this goes on forever, perhaps to the last recorded syllable.”

```
Have you seen Ms.~Jones?
```

Have you seen Ms. Jones?

```
Prof.~Smith and Dr.~Gold flew from
Halifax N.~S. to Montr\'eal, P.~Q. via Moncton, N.~B.
```

Prof. Smith and Dr. Gold flew from Halifax N. S. to Montréal, P. Q. via Moncton, N. B.

```
\line{left end \hfil left tackle \hfil left guard \hfil centre\hfil
right guard \hfil right tackle \hfil right end}
```

left end left tackle left guard centre right guard right tackle right end

---

```
\line{left \hfil \hfil right-centre\hfil right}
```

left right-centre right

```
\line{\hskip 1 in ONE \hfil TWO \hfil THREE}
```

ONE TWO THREE

```
i{f}f if{}f if{f}
```

iff iff iff

```
I started with roman type {\it switched to italic type}, and
returned to roman type.
```

I started with roman type *switched to italic type*, and returned to roman type.

```
$C(n,r) = n!/(r!\,(n-r)!)$
```

$C(n,r) = n!/(r!\,(n-r)!)$

```
$a+b=c-d=xy=w/z$
$$a+b=c-d=xy=w/z$$
```

$a+b=c-d=xy=w/z$

$$a+b=c-d=xy=w/z$$

```
$(fg)' = f'g + fg'$
$$(fg)' = f'g + fg'$$
```

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

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

```
$\alpha\beta=\gamma+\delta$
$$\alpha\beta=\gamma+\delta$$
```

$\alpha\beta=\gamma+\delta$

$$\alpha\beta=\gamma+\delta$$

```
$\Gamma(n) = (n-1)!$
$$\Gamma(n) = (n-1)!$$
```

$\Gamma(n) = (n-1)!$

$$\Gamma(n) = (n-1)!$$

```
$x\wedge (y\vee z) = (x\wedge y) \vee (x\wedge z)$
```

$x\wedge (y\vee z) = (x\wedge y) \vee (x\wedge z)$

```
$2+4+6+\cdots +2n = n(n+1)$
```

$2+4+6+\cdots +2n = n(n+1)$

```
$\vec x\cdot \vec y = 0$ if and only if $\vec x \perp \vec y$.
```

$\vec x\cdot \vec y = 0$ if and only if $\vec x \perp \vec y$.

```
$\vec x\cdot \vec y \not= 0$ if and only if $\vec x \not\perp \vec y$.
```

$\vec x\cdot \vec y \not= 0$ if and only if $\vec x \not\perp \vec y$.

```
$(\forall x\in \Re)(\exists y\in\Re)$ $y>x$.
```

$(\forall x\in \Re)(\exists y\in\Re)$ $y>x$.

```
${a+b\over c}\quad {a\over b+c}\quad {1\over a+b+c} \not= {1\over a}+
{1\over b}+{1\over c}$.
```

${a+b\over c}\quad {a\over b+c}\quad {1\over a+b+c} \not= {1\over a}+{1\over b}+{1\over c}$.

```
What are the points where ${\partial \over \partial x} f(x,y) = {\partial \over
\partial y} f(x,y) = 0$?
```

What are the points where ${\partial \over \partial x} f(x,y) = {\partial \over \partial y} f(x,y) = 0$?

```
$e^x \quad e^{-x} \quad e^{i\pi}+1=0 \quad x_0 \quad x_0^2
\quad {x_0}^2 \quad 2^{x^x}$.
```

$e^x \quad e^{-x} \quad e^{i\pi}+1=0 \quad x_0 \quad x_0^2 \quad {x_0}^2 \quad 2^{x^x}$.

```
$\nabla^2 f(x,y) = {\partial^2 f \over\partial x^2}+ {\partial^2 f \over
\partial y^2}$.
```

$\nabla^2 f(x,y) = {\partial^2 f \over\partial x^2}+ {\partial^2 f \over \partial y^2}$.

```
$\lim_{x\to 0} (1+x)^{1\over x}=e$.
```

$\lim_{x\to 0} (1+x)^{1\over x}=e$.

```
The cardinality of $(-\infty, \infty)$ is $\aleph_1$.
```

The cardinality of $(-\infty, \infty)$ is $\aleph_1$.

```
$\lim_{x\to {0^+}} x^x = 1$.
```

$\lim_{x\to {0^+}} x^x = 1$.

```
$\int_0^1 3x^2\,dx = 1$.
```

$\int_0^1 3x^2\,dx = 1$.

```
$\sqrt2 \quad \sqrt {x+y\over x-y} \quad \root 3 \of {10}$ \quad $e^{\sqrt x}$.
```

$\sqrt2 \quad \sqrt {x+y\over x-y} \quad \root 3 \of {10}$ $\quad$ $e^{\sqrt x}$.

```
$\|x\| = \sqrt{x\cdot x}$.
```

$\|x\| = \sqrt{x\cdot x}$.

```
$\phi(t) = {1 \over \sqrt{2\pi}} \int_0^t e^{-x^2/2}\,dx$.
```

$\phi(t) = {1 \over \sqrt{2\pi}} \int_0^t e^{-x^2/2}\,dx$.

```
$\underline x \quad \overline y \quad \underline{\overline{x+y}}$.
```

$\underline x \quad \overline y \quad \underline{\overline{x+y}}$.

```
$\bigl \lceil \lfloor x \rfloor \bigr \rceil \leq \bigl \lfloor \lceil x \rceil
\bigr \rfloor$.
```

$\bigl \lceil \lfloor x \rfloor \bigr \rceil \leq \bigl \lfloor \lceil x \rceil \bigr \rfloor$.

```
$\sin(2\theta) = 2\sin\theta\cos\theta
\quad \cos(2\theta) = 2\cos^2\theta - 1 $.
```

$\sin(2\theta) = 2\sin\theta\cos\theta \quad \cos(2\theta) = 2\cos^2\theta - 1$.

```
$$\int \csc^2x\, dx = -\cot x+ C
\qquad \lim_{\alpha\to 0} {\sin\alpha \over \alpha} = 1
\qquad \lim_{\alpha\to \infty} {\sin\alpha \over \alpha} = 0.$$
```

$$\int \csc^2 x\, dx = -\cot x + C \qquad \lim_{\alpha\to 0} {\sin\alpha \over \alpha} = 1 \qquad \lim_{\alpha\to \infty} {\sin\alpha \over \alpha} = 0.$$

```
$$\tan(2\theta) = {2\tan\theta \over 1-\tan^2\theta}.$$
```

$$\tan(2\theta) = \frac{2\tan\theta}{1-\tan^2\theta}.$$

```
\proclaim Theorem (Euclid). There exist an infinite number of primes.
```

Theorem (Euclid). *There exist an infinite number of primes.*

```
\proclaim Proposition 1.
$\root n \of {\prod_{i=1}^n X_i} \leq
{1 \over n} \sum_{i=1}^n X_i$ with equality if and only if $X_1=\cdots=X_n$.
```

Proposition 1. $\sqrt[n]{\prod_{i=1}^n X_i} \leq \frac{1}{n} \sum_{i=1}^n X_i$ *with equality if and only if* $X_1=\cdots=X_n$.

```
$$ I_4 = \pmatrix{
1 &0 &0 &0 \cr
0 &1 &0 &0 \cr
0 &0 &1 &0 \cr
0 &0 &0 &1 \cr}$$
```

$$I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
$$ |x| = \left\{ \matrix{
x & x \ge 0 \cr
-x & x \le 0 \cr} \right.$$
```

$$|x| = \left\{ \matrix{ x & x \ge 0 \cr -x & x \le 0 \cr} \right.$$

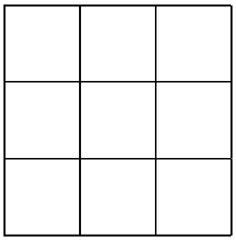
```
\settabs \+ \hskip 2 in & \hskip .75in & \hskip 1cm& \cr
\+ &Plums &\hfill\$1&.22 \cr
\+ &Coffee &\hfill1&.78 \cr
\+ &Granola &\hfill1&.98 \cr
\+ &Mushrooms & &.63 \cr
\+ &{Kiwi fruit} & &.39 \cr
\+ &{Orange juice} &\hfill1&.09 \cr
\+ &Tuna &\hfill1&.29 \cr
\+ &Zucchini & &.64 \cr
\+ &Grapes &\hfill1&.69 \cr
\+ &{Smoked beef} & &.75 \cr
\+ &Broccoli &\hfill\underbar{\ \ 1}&\underbar{.09} \cr
\+ &Total &\hfill \$12&.55 \cr
```

Plums	\$1.22
Coffee	1.78
Granola	1.98
Mushrooms	.63
Kiwi fruit	.39
Orange juice	1.09
Tuna	1.29
Zucchini	.64
Grapes	1.69
Smoked beef	.75
Broccoli	1.09
Total	\$12.55

```
\settabs \+ \hskip 4.5 in & \cr
\+Getting Started \dotfill &1 \cr
\+All Characters Great and Small \dotfill &9 \cr
```

Getting Started 1
All Characters Great and Small 9

```
\settabs \+ \hskip 1cm&\hskip 1 cm&\hskip 1 cm& \cr
\moveright 2 in
\vbox{
\hrule width 3 cm
\+ \vrule height 1 cm & \vrule height 1 cm & \vrule height 1 cm
& \vrule height 1 cm \cr
\hrule width 3 cm
\+ \vrule height 1 cm & \vrule height 1 cm & \vrule height 1 cm
& \vrule height 1 cm \cr
\hrule width 3 cm
\+ \vrule height 1 cm & \vrule height 1 cm & \vrule height 1 cm
& \vrule height 1 cm \cr
\hrule width 3 cm
}
```



```
\def\boxtext#1{%
\vbox{%
\hrule
\hbox{\strut \vrule{} #1 \vrule}%
\hrule
}%
}
\moveright 2 in \vbox{\offinterlineskip
```

```
\hbox{\boxtext{6}\boxtext{1}\boxtext {8}}
\hbox{\boxtext{7}\boxtext{5}\boxtext{3}}
\hbox{\boxtext{2}\boxtext{9}\boxtext{4}}
}
```

6	1	8
7	5	3
2	9	4