Doing a TEXjob

TEX is not a word processor. Rather, you use your favorite word processor or file editor to prepare files of instructions for TEX. By and large, it doesn't matter what word processor you use. Since braces turn out to be important in TEX, you are better off using a file editor like **`vi`** that will allow you to check that every closing brace (or parenthesis or bracket) matches the opening brace (or parenthesis or bracket) you want it to match. It is also important that the word processor leave no funny control characters in the file, as these might confuse TEX.

How such an instruction file (traditionally carrying the suffix `.tex`) is converted into pages of beautiful print need not concern you (though you will come to admire the cleverness that must have gone into the process). But you have to find out how, on your computer system, you submit that instruction file to TEX. Usually, a command like `tex myfile` suffices if `myfile.tex` is the file containing the instructions. Your system should also allow you to have the file worked on by TEX without any final printing, since that is the way to find out mistakes.

ordinary text

You type ordinary text as you would in any word processor, except that there are *fewer* things to pay attention to.

(1) As you can see, it doesn't matter how many spaces you leave between words or after punctuation. TEX will choose an even and optimal spacing and leave the right number of spaces after punctuation.

(2) As you can see, it doesn't matter where you break lines. TEX will choose optimal line breaks (if that is possible; it will let you know when it cannot; actually, I tried hard to force such a situation in the next item without trickery).

(3) In particular, you (usually) need not pay attention to hyphenation. TEX is completely familiar with all the rules and applies them automatically, if need be. (If you are eager to supply a particular hyphenation, indicate the place by typing `\-` there; e.g., `surg\-e\-ry`.)

You indicate the end of a paragraph by leaving a blank line (or two or three if you want to be generous; again, it doesn't matter how many you leave).

You can also indicate the beginning of a new paragraph by typing `\par`.

Special characters

You type all characters great and small on your keyboard, except that some (altogether ten) non-letters have special meaning for TEX and therefore must be typed in a special way if they are actually to appear in the printed text. Here is the list of these special characters:

$$\backslash \quad \{ \quad \} \quad \$ \quad \& \quad \# \quad \hat{} \quad \_ \quad \% \quad \tilde{}$$

You notice that you get most of these symbols to print by typing first the backslash `\`. For some, this will work only if this is (part of a text) enclosed in dollar signs (of which much more later). The backslash itself is so important that TEX requires you to type out the word `backslash` after a `\` (and encase the whole thing in dollar signs) if you want to have the backslash printed.

The backslash serves as an **escape character**, i.e., as a message to TEX that what immediately follows the backslash is to be taken as a command (rather than as text to be typeset).

In particular, you get all the symbols and signs *not* on your keyboard with the aid of such **command**s. See Sam Bent's TEX Reference Card in the Appendix for a complete, ordered listing.

odds and ends

TEX will automatically leave more space after a period at the end of a sentence than between words in a sentence. But since TEX has no way of understanding what you write, it has to make a guess at what is 'a period at the end of a sentence'. It guesses that it is any period followed by one or more blanks as long as it is not preceded by a capital letter, as in D. E. Knuth. Note that TEX provides an ordinary interword space after each of the two initials, and a larger, intersentence, space after the final period.

This rule of thumb causes difficulty when a sentence ends in a capital letter, such as this ONE. See? You can overcome it by inserting the **do nothing** command `\null` just prior to the PERIOD. There. That does it.

This rule of thumb also causes difficulty with abbreviations, such as refs. to comp.lit. or phys.ed. courses. You overcome this by following such abbreviation periods by a **forced blank**, e.g., as in refs. to comp.lit. or phys.ed. courses. See the difference?

This rule of thumb causes difficulty with the unhappy habit and unfortunate standard rule of 'putting the period at the end of a quote *inside* the quote.' The remedy is simple: 'Put the period outside'. But if you must keep it inside, "follow the quote with two forced blanks." There.

Note that the beginning quotes in the preceding paragraph were typed differently from what you might have expected. Just to get used to this, also type the closing double quote as " even though " will give the same thing.

simple math

All math within text is enclosed within dollar signs, even when it is just one letter or symbol, such as a, b, or c, and certainly for things like $E=mc^2$, or $x_i+y_i={3\over4}$, or a^{b^c}, or x_1^α, or or $\int_{-1}^1 x\,dx=0$, or $\sum_{i=0}^n c^i=(c^{n+1}-1)/(c-1)$ for all $n\ge0$.

It is worth studying these examples in some detail. I have not left any spaces around symbols; TEX takes care of that, usually. But TEX cannot understand math, so sometimes you may have to control spacing, as I have done in the integral, by putting a bit of space (via `\,`) between the x and the dx. Also, one must use braces to indicate the extent of subscripts and superscripts, but only if they involve more than one character. Braces are also required to control what `\over` puts on top and below the bar.

For a complete, ordered list of all the available math symbols, see Sam Bent's TEX Reference Card in the appendix.

The inexperienced typist will bemoan having to type all those dollar signs. Although I do touch-typing, sort of, I have found it more convenient to type the dollar sign (and some other signs) piano-fashion, i.e., by hitting appropriate keys simultaneously with the same hand; in this example, it's the left hand, with the little finger hitting the shift key while the middle finger simultaneously hits the 4/\$ key.

display math

Displayed math is enclosed within **double** dollar signs. Here are some of the earlier math examples, but in double dollar signs:

$$E = mc^2, or x_i + y_i = \frac{3}{4}, or a^{b^c}, or x_1^\alpha,$$

or

$$\int_{-1}^{1} x\,dx = 0,$$

or

$$\sum_{i=0}^{n} c^i = (c^{n+1} - 1)/(c - 1), for all n \geq 0.$$

You'll notice that some of the display has become more expansive. For example, the summation sign is bigger, and its limits are above and below it, rather than to the right of it. That's nice since TEX takes care of such things; you type it the same way, whether in math mode or in display mode, and TEX makes the appropriate adjustments. You'll also notice that the last comma in the first and second display, and the period in the third display, are now typed before the dollar sign(s) rather than after. Can you guess what would have happened otherwise?

On the other hand, notice what happened to the ordinary word "or" and the text "for all". TEX treats them, not as words, but as a sequence of math symbols, hence puts them into italics, gives them a funny spacing and ignores entirely the interword space. To have them typeset as ordinary text, you need to switch temporarily to that font, as in the following:

$$\sum_{i=0}^{n} c^i = (c^{n+1} - 1)/(c - 1), \mathrm{forall} n \geq 0.$$

This now uses ordinary (roman) type for these two words, but still isn't right since, being inside display mode, the interword spacing is ignored. So you have to enforce it by using the **forced blank**:

$$\sum_{i=0}^{n} c^i = (c^{n+1} - 1)/(c - 1), \mathrm{for\ all}\ n \geq 0.$$

This is still not quite right since it is nicer to set off that last quantifier from the actual formula. This you do by inserting some (standard) space that TEX provides with the commands `\quad` and `\qquad`. `\qquad` is the right one here:

$$\sum_{i=0}^{n} c^i = (c^{n+1} - 1)/(c - 1), \qquad \mathrm{for\ all}\ n \geq 0.$$

alignment

TEX really begins to pay off when it comes to the alignment of text. Here are some standard *examples*:

$$H(x) := \begin{cases} x & \text{if } 0 \le x \le 1; \\ 2-x & \text{if } 1 \le x \le 2; \\ 0 & \text{otherwise.} \end{cases}$$

This is an example of the `\cases` statement. The description of each line has two **fields**. The first field goes from the 'beginning' of the description to the **ampersand** `&`; it is typeset automatically as *math*. The second field goes from the ampersand to the **carriage return** `\cr`, and it is typeset automatically as *text*; hence any math in it must be embraced by dollar signs. The description of the first line begins right at the opening brace of the `\cases` statement. The description of subsequent lines begins right after the `\cr` of the preceding line. (It doesn't matter how many actual lines you use in your `.tex` file to describe the lines to be displayed in the cases statement.)

If you prefer somewhat more space between the cases, use `\noalign`, as in this modification, which also shows how you could get the material in the first fields centered:

$$H(x) := \left\{ \begin{array}{cl} x & \text{if } 0 \le x \le 1; \\[1ex] 2-x & \text{if } 1 \le x \le 2; \\[1ex] 0 & \text{otherwise.} \end{array} \right.$$

The next example shows how to align several lines of displayed equations. The description of each line is terminated again by the carriage return, and, in each description, the point of alignment is again marked by an ampersand.

$$\begin{aligned} M{*}'f &= M{*}(M_|{*}c) \\ &= M{*}(c{*}M_|) \\ &= (M{*}c){*}M_| = f{*}'M. \end{aligned}$$

Incidentally, the asterisks have all been em'brace'd here to prevent TEX from embedding them in extra space (a thing it would do to any symbol it recognizes as a binary operation sign).

Many math alignments can be handled by using **matrices**. Here is the standard example:

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \tag{14}$$

in which you also see various ways of entering the equivalent of **etc** into math formulas as well as an equation number. The macro `\pmatrix` provides its own fences. If you prefer some other kind, you use `\matrix` together with `\left` and `\right`, as in

$$\begin{bmatrix} \mathrm{A} & \mathcal{B} & C \\ \mathbf{D} & F & \Gamma \end{bmatrix}, \tag{14$'$}$$

which also illustrates that you can treat each matrix entry as if it were typeset without regards to any other entry (which it is).

Note that, once again, the ampersand `&` indicates the alignment points while the carriage return `\cr` indicates the end of a line (description).

The use of **over-** and **underbrace** is illustrated nicely here:

$$\{\underbrace{\overbrace{a,\ldots,a}^{h\ a'\text{s}},\overbrace{b,\ldots,b}^{k\ b'\text{s}}}_{h+k\ \text{elements}}\}.$$

Actually, this looks a bit crooked. To get it right, add a `\mathstrut` to each group to be 'overbraced' in order to give them the same height:

$$\{\underbrace{\overbrace{\mathstrut a,\ldots,a}^{h\ a'\text{s}},\overbrace{\mathstrut b,\ldots,b}^{k\ b'\text{s}}}_{h+k\ \text{elements}}\}.$$

This is still not quite right, since we would want apostrophes rather than primes in the overbrace material, but 'twill serve.

The next example shows the general-purpose and very useful `\halign` command in action:

$$\begin{array}{llcccc}
\text{for } f \in \Pi_M, & M{*'}f = f{*'}M & = & f\sum_{j\in\mathbb{Z}^d} M(j) & - & \sum_{j\in\mathbb{Z}^d}(f - f(\cdot - j))M(j) \\
 & & \in & f & + & \Pi_{<\deg f}
\end{array}$$

Here, too, we describe how each of the displayed lines should look. As before, **field**s are separated by ampersands `&`'s, and line descriptions by carriage returns `\cr`'s. But the first line description is special. It doesn't describe a particular line; rather, it is a **template** for the actual lines, i.e., it describes a general **pattern** to be followed by all (subsequent) lines. This means that each field in the first line description tells how the **corresponding** field in the actual lines should be handled, with the **sharp** sign `#` indicating just where in that field the actual material in each line for that field is to appear.

In the above example, the description of the first field is simplest. It contains the sharp sign `#` and nothing else. The second field starts with some space and then has the sharp `#` embraced by dollar signs. Hence, for each line, the second field will start with the same specified amount of space followed by the actual material from the second field, but typeset in *math* mode. The fourth field (it is part of the trickiness of `\halign` to keep proper track of which field is which) uses `\hfil` to put some **variable** space around the material for that field, which, in effect, **centers** that material within the field. It also embraces the material with dollar-signs, hence will do it in math mode. The next field is entirely in math-mode, but embraces the sharp sign with a little space (indicated by `\;` which only works in math-mode), and so puts that space around the material for that field in each actual line. And so on.

The only thing not quite right about the above display is lack of centering and lack of proper surrounding space. But that can be arranged, using `\noalign` to put some space between the lines, and things called `\hbox` and `\vbox` to center it all, as you see here:

$$\begin{array}{lrcl}\text{for } f \in \Pi_M, & M{*}'f = f{*}'M & = & f\sum_{j\in\mathbb{Z}^d} M(j) - \sum_{j\in\mathbb{Z}^d}(f - f(\cdot - j))M(j) \\ & & \in & f \qquad + \qquad \Pi_{<\deg f}\end{array}$$

I have to admit that getting an alignment of such complexity right takes some doing and some patience, and some struggle with TEX's often inscrutable error messages.

This particular example might actually be easier to handle with the `\matrix` command, which gives:

$$\begin{matrix}\text{for } f \in \Pi_M, & M{*}'f = f{*}'M & = & f\sum_{j\in\mathbb{Z}^d} M(j) & - & \sum_{j\in\mathbb{Z}^d}\big(f - f(\cdot - j)\big)M(j) \\ & & \in & f & + & \Pi_{<\deg f}\end{matrix}$$

Here is a simple use of `\halign` which also demonstrates the use of `\hbox` and `\vbox`, well, of its cousin `\vtop` (see the section on boxes for details).

Michael G. Crandall	and	Pierre-Louis Lions
Department of Mathematics		Ceremade
University of Wisconsin-Madison		Université Paris-Dauphine
Madison WI 53706		Place de Lattre de Tassigny
		75775 Paris Cedex 16

The function served here by `\halign` is to permit the name-and-address information to be entered in line, separated only by the carriage returns, and to make sure that all these lines are left-adjusted, and to supply a box that is exactly large enough to contain all that information, hence makes it possible to align these two boxes properly on the page. But it makes it also very easy to **center** the information instead, by adding another `\hfill` to the template in `\halign`:

Michael G. Crandall	and	Pierre-Louis Lions
Department of Mathematics		Ceremade
University of Wisconsin-Madison		Université Paris-Dauphine
Madison WI 53706		Place de Lattre de Tassigny
		75775 Paris Cedex 16

Table construction is also easily done with the aid of `\halign`. Here is a simple **table**.

Table 1

x	$\sin(x)$
0	0
$\pi/2$	1

Note the use of rules to draw lines. Note in particular how that vertical line is drawn for each line with the aid of the second field in the pattern. Note that the pattern description for that second field must contain a sharp even though you have no intention of ever putting anything into that field (other than the `\vrule` specified by the pattern).

But it's a miserable table otherwise. The heading line and the first line in the table are right on top of each other, yet there's some skip between the two lines in the table, and the columns are just big enough to contain the widest item in each.

We fix the crowding of the lines by putting a `\strut` into the pattern (i.e., a `\vrule` of zero width but of sufficient height and depth that the resulting lines of type can be allowed to be contiguous without any interlineskip and still look well separated). This will also fix the problem with the skip between the lines; for, with the `\strut` making all lines a little bit larger, we can turn off the interlineskip inside the box that is to contain the table by saying `\offinterlineskip`. Finally, we can make the columns a little bit wider by adding some forced blanks in the patterns. Here is the result:

Table 1

x	$\sin(x)$
0	0
$\pi/2$	1

For the more demanding consumer, here is a more sophisticated table in which repeated `\hrule`s have given some lines greater thickness, and in which some entries actually go across several columns.

Table 2

$\theta = 2\,\mathrm{deg}$			Flight data		Computation	
R^*	τ^*	b	C_{lrm}	C_{lsm}	C_{lrm}	C_{lsm}
10.0	.091	4.32	-.025	.020	-.02375	.02437
20.0	.087	5.20	-.040	.040	-.02984	.03571
45.2	.123	4.23	-.055	.050	-.04454	.04714

Note the use of `\hbox to \hsize` and `\hfill` to center the table. By the way, in typing in the description, I made extensive use of the fact that, in **vi**, it is very easy to duplicate and repeatedly use a word, or phrase, or entire line, and that **vi** will automatically show me matching braces, so I would be sure which grouping I was closing. As a result, the table came out perfect the first time I ran TEX on it, – except that I had typed `Fight` rather than `Flight`, and had typed `C_{\rm}` rather than `C_{lrm}`. Immerhin.

TEX also makes available the main tool for vertical alignment on a typewriter, the **tab key**. I have never used it since I find `\halign` so much more versatile and I don't have to count spaces beforehand. (In fact, TEX implements tabbing with the aid of `\halign`.) Tabbing is useful for horizontal alignments that run over many pages since, in contrast to `\halign`, tabbed material can be typeset line by line. By contrast, `\halign` must first look at all the columns before it can settle column width and spaces between columns.

Finally, there are occasions when you want to locate material just so. For example, you might have a graph you wish to label. (TEX makes available the macro `\special` which allows you to include, e.g., graphic material right with the TEXed material. Details depend on the printer you are using and on the particular software that converts the dvi file TEX produces into a file that makes sense to your printer.) For this, look for the macros `\gridbox` and `\point` in the section headed 'including and labelling figures', which are also useful for precise placement without any figure involved.

We now use `\vfill\eject` to generate a clean page break.

The size of the overall area on a page to be filled with print can be controlled by setting `\hsize` and `\vsize`.

The placement of this rectangle of printed text is controlled by setting `\hoffset` and `\voffset`.

The space between paragraphs is controlled by `\parskip`.

The amount of indentation is controlled by `\parindent`.

The amount of space between lines can be controlled by setting `\baselineskip`, as I have just done. All these parameters have default values. You set them only to change those default values. Their new values are used as soon as you set them and until you come to the end of the current grouping. After that, they revert to what they were before entering that grouping.

> If you want to restrict these changes to a particular part of the text, put that text into braces that also embrace those changes, as I have done for the preceding three paragraphs (and also for the entire material on this page).
>
> > The preceding paragraph has had its margins widened by use of `\narrower`. Also, the indentation for the present paragraph has been suppressed by a `\noindent`, and the paragraph made yet narrower by a `\narrower\narrower`. But, in each case, the effect of the narrowing command has been restricted to the paragraph by embracing the *completed* paragraph.

It is important to note that parameters that only affect entire paragraphs (such as `\baselineskip` or `\narrrower`) only affect those paragraphs that are completed before the end of the grouping within which they occur is reached. For *example*, leaving off the `\par` near the end of the description of the preceding paragraph would have prevented the `\narrower\narrower` at its beginning from taking effect.

`\parindent` has just been made quite small. Also, `\hang` has been used to indent the entire paragraph. The amount of indenting is determined by the current value of `\parindent`.

• You can get the same effect with `\item`, except that `\item` gives you the opportunity to put something to the left of the indent on the first line, as I have done here.

Note the effects of

a `\leftline`

a `\centerline`

and a `\rightline`

and of a `\line` with some `\hfill`

On this page, you see some heading and some footing. Also, the page number also occurs at the upper right corner. All of this is the result of setting two built-in macros called `\headline` and `\footline`. As you read the definitions, you will appreciate the possibility of having the left-page headline differ from the right-page headline, or of having the page number appear at the bottom center as long as it is a roman numeral (indicated by having `\pageno` negative), and at the outer upper page corner otherwise.

For this, you need to know that `\folio` gives you the decimal digits of the current page number (i.e., the value of `\pageno`) in the current font, in case that number is nonnegative, and gives you the roman numerals of the negative of `\pageno` otherwise. That works out fine for the title matter in a book (and since the Romans didn't have a zero). You also want to know that `\pageno` is something that TEX will increase by one every time it finishes a page (or decrease by one if it was negative). But, if you don't like the current number and rather have it be the number 1937, you could say `\pageno=1937` and that would take care of it.

In fact, if you don't want page numbers at all, say `\nopagenumbers`. This is the same as saying `\footline{\hfil}`, but makes the purpose clearer.

I'll restore all of this (including `\pageno`) back to what it was at the beginning of the next page. Look there for how it is done.

More complicated schemes are available. E.g., it is possible to have the footline contain the latest marked expression in the text prior to the page break. But for such things, you had better consult the book.

It is also good to know how to do a

Table of Contents

commands, macros, definitions

TEX commands (also called **macro**s, or **definition**s) begin with a backslash and are followed either by exactly one non-letter, or else by one or more letters. There is no limitation on how many letters such a command 'word' is made up from. TEX will read the letters until it comes to a non-letter and then take all the letters read together as being the command word. If that terminating non-letter is a blank (or space), then it (and all blanks following it) will be lost.

For *example*, you have been seeing the TEX logo. The instruction for its typing is such a command, but you see the command being followed by another backslash and a space. The latter command is of the non-letter variety, the non-letter being the blank. It tells TEXto put a blank space in. Can you tell why that is necessary? Look also for earlier examples, in which the command for the logo is *not* followed by the blank command.

Available commands are either **primitive** commands, or else **macros**, i.e., an *abbreviation* for a sequence of commands. The description of the heading for the next section contains an *example* of such a macro, defined just prior to it to combine the three commands used in the description of earlier headings.

Sam Bent's TEX Reference Card in the Appendix contains a complete, ordered listing of all the primitive commands as well as the plainTEXsupplied macros. But the fact that you can make up your own macros to suit your own purposes is one of the major attractions of TEX.

For *example*, I have collected a file consisting entirely of macro definitions. The file happens to be called `format.tex`. To make certain that these macros are available to me when I work on a `.tex` file, I start the `.tex` file with the command `\input format`. This instructs TEX to become familiar with all those macros in the file `format.tex` before starting to work on the instructions in the `.tex` file. The full file is available by anonymous ftp at `ftp.cs.wisc.edu/Approx` and includes definitions like

`\def\ga{\alpha}` for **g**reek **a**lpha; or

`\def\gO{\Omega}` for **g**reek Capital **O**mega, so I only have to type `$\ga$ and $\gO$` to get 'α and Ω'; or

`\def\bs{\backslash}` for the printed **b**ack**s**slash, so I only have to type `\bs` to get a backslash in the text (instead of the long word `backslash`).

(Please admire the actual instructions for the typing of the last item. The great convenience brought on by making ten characters special brings much inconvience when you actually want to print those characters. (The difficulty is compounded by writing it all in typewriter font.) Fortunately, you seldom need to type these special characters. My trip-up is usually the dollar or percent sign in a letter, or the sharp sign in math-mode (where it denotes cardinality), or the ampersand in a list of names or a reference.)

Why would use of the macro `\deadly` defined by `\def\deadly{\deadly}` be deadly?

commands with arguments

The heading of this section is put in with the aid of the macro `\heading`. Have a look at it. It uses an argument, namely the material that is to appear in the heading. Now look at its definition, just above its use. You can tell from the definition that it is intended to use an argument because you see the symbols `#1` right between the name of the macro and the opening brace of its 'body'. Also, you see within the body the symbols `#1` repeated right at the spot where the material that makes up the argument is to be placed.

Here is how such a macro is understood by TeX. After TeX has read the macro's name (TeX will know that the name is `heading` and not something longer because the character following the `g` in `\heading` either is a blank or else a non-letter), TeX will look up the definition, find that it requires an argument and now look for it. It expects to find an opening brace as the next character. If that is indeed the case, it will take everything between this opening brace and the *corresponding* closing brace as the argument. In this particular example, this means that everything between that opening and corresponding closing brace will end up centered, with the centerline preceded and followed by some vertical space.

TeX also allows for a shortcut. If an argument consists of just one character, then it is not necessary to enclose it in braces. In other words, if TeX does not find an opening brace as the next character, then it takes that next character as the whole argument.

A macro may have up to nine separate arguments. In the definition of the macro, they are listed, between the macro's name and the opening brace of the macro's 'body', as `#1#2...`. They should also (but don't have to) appear at least once inside the macro's body, exactly at the spot at which the material that forms the argument is to appear. The intent is to have the macro provide a template with certain places left open, to be filled in with particulars when the macro is actually used.

At first glance, this list of `#1#2#3...` between macro name and body looks a bit silly; why not simply say `5` if five arguments are expected? But this is yet another cleverness of TeX. For, you are permitted to put any one character after each of those numbers, for example, `#1,#2|#3#4/...`, and these very characters are used later by TeX to decide when one argument ends and the next one begins. For example, I have a friend (not a piano player) who hates typing dollar signs. He has a macro that he defined as follows:

```
\def\m#1:{$#1$}
```

If he has to type something in math-mode, e.g. $\alpha = 1/\gamma$, he would type

```
\m\alpha=1/\gamma:
```

TeX will pick up the `m` as the name of a macro, look it up and find that it has one argument and that the extent of the argument is all the stuff following that `\m` until it comes across a colon. So it picks up all that stuff and, following instructions, puts it between dollar signs and then processes it in the usual way. The colon itself will not be printed; it was used up as the **delimiter** or **terminator** of the first (and only) argument of the macro.

What is he going to do when his math-stuff contains a colon, e.g., he wants to type $\{x : f(x) = 0\}$? Then he hides that colon in braces! I.e., he types

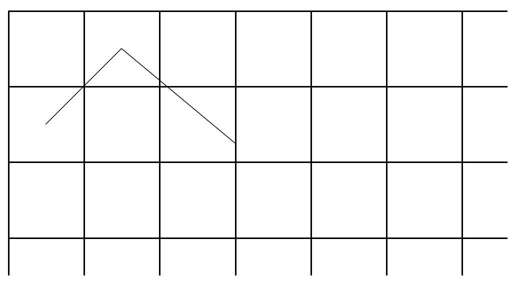
```
\m\{x{:} f(x)=0\}:
```

For, as TeX reads the stuff following `\m`, it skips over any groupings, i.e. over any stuff between braces in its search for the delimiting or terminating colon.

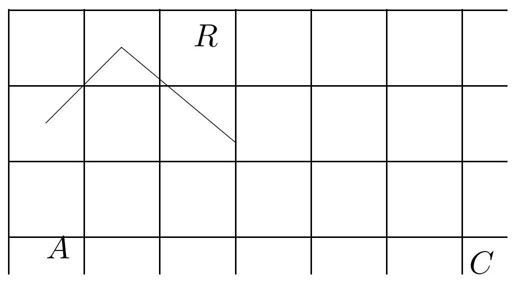
including and labelling figures

Here are the definitions of `\gridbox` and `\point`. These macros are useful for the precise placement of material. I illustrate their use in labelling a simple graph. The actual graph is specified with the aid of PostScript, the same language that is used on many Laser writers to print TEX. Different printers or different converters from dvi-file to printer file would require different statements to get the graphic combined with text. A popular (free) means for placing figures into TEX text is `\epsf`.

We start with the graphic, put into the gridbox, with the grid drawn, to help us later on to place the labels.



Next we take a stab at placing the labels.



That looks ok, except that the R should be a little bit more to right and down, say .35cm down and .4 cm to the right. So we change these 'coordinates'. We also get rid of the grid (which we can do by saying `\showgridfalse` or by saying `\gridwidth=0pt`), center the whole figure, and put a Figure description underneath.

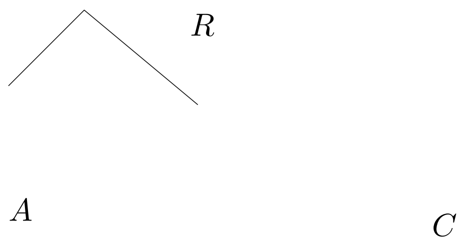


Figure 3. A labelled graph

There, that wasn't so bad. – One would usually work on this in a separate file and only insert the finished material appropriately, perhaps using `\midinsert` or `\topinsert`.

At times, it is convenient to pack it all in. This is one of those times, I think. (You might try to modify the macro `\boxit` used here to get a double box line.)

sequencing equations and other items

(1) Here is the macro `\label` which is useful for an automatic sequencing of equations and other items. It is so simple that it wouldn't be hard for you to modify them to fit the particular needs of a particular paper.

(2) `\label` will increment the value of `\equationno` by one and then print it out, and will enclose it in parentheses if called in display-math mode, as in the following.

(3) $$e = mc^{1/2}$$

(4) In order to refer to these numbers later, you give them names. For example, I gave the name `\listequ` to the number that starts the paragraph (2) (**check how I typed the preceding number**) by saying `\label\listequ` there.

(38) You can always change `\equationno` to any value you like. For example, I just changed it to 37 by typing `\equationno=37` prior to this paragraph.

“memory (39) If you find that you have trouble remembering the names you gave to earlier items (and you don't want to search for it in the `.tex` file), you can say `\showlabeltrue` and from then on the name you used will appear, quite small, nearby in the left margin of the printed document.

(40) Guess what you type to turn off this feature?

(41), (42), ...

(∞) Finally, you may wish to refer to the very equation- or item-numbers named in this `.tex` file in other files. You would want to write them as definitions into a file ⟨*filename*⟩ which you would then `\input` ⟨*filename*⟩ in the other file. For this to work here, you would first say something like

```
\rememberchaptertrue \newwrite\chpaux
\immediate\openout\chpaux=⟨filename⟩
```

when you make all those other definitions in this section. The macro `\label` then takes care of the rest.

($\infty + 1$) Actually, the way it is set up in this section, the name saved is given the suffix `\chapterno`, with the assumption that `\chapterno` is some word identifying this particular file. Furthermore, the number saved is prefixed by `\chapterno.` . This means that you can safely use names in the present file without worrying about the fact that you might use the same name for something else in the other file. Enough already.

boxes

A box has three dimensions: **height**, **width**, and **depth**. These measure the extent of the box in reference to its **reference point**. **height** is its height above, **width** its extent to the right of, and **depth** its depth below, its reference point. This reference point serves as a means of alignment when boxes are put together. See Knuth, page 63, for a beautiful picture.

Boxes are put together to make (more complicated) boxes. They can be put together vertically or horizontally. Either way, they are lined up by their reference points. In a vertical list, each box is stacked below its predecessor so as to align their reference points, while, in a horizontal list, each box is placed to the right of its predecessor so as to align their reference points.

Once a list has been used to make a box, the list is forgotten, and only the box remains. It is just a box. You can't tell whether the box was made by stringing boxes together vertically or horizontally. Neither can you take it apart again.

TeX is in vertical or horizontal mode, depending on whether it is putting together a vertical or a horizontal list of boxes.

TeX switches from one mode to the other when it comes across an item that doesn't allow it to go on in the present mode.

In the most simple setup, TeX is putting together letters and blanks to make up a line. In terms of boxes, it is making up a horizontal list. The resulting box is a line of type.

Even when TeX is in vertical mode, if it comes across a letter, a (forced) blank or other character, it will temporarily suspend vertical mode and start making a line of type. As this is the first line (of possibly several lines), it will begin with an indent (whose length is specified by `\parindent`). Subsequent items in the list all will contribute to that line of type and possibly further lines of type, just as if TeX had been asked to make up a page.

A list of such boxes, each a line of type, forms the basic vertical list. The resulting box is a page of text.

The simplest boxes are those consisting of just one letter or other character. Their dimensions are determined by the font designer. This information is available to TeX in appropriate `.fnt` files.

TeX makes up automatically the boxes that represent a line of type and the boxes that represent a page, and the sizes of these boxes are determined by default or by certain parameters that are set by default but could be set by the user.

A box made up automatically from a horizontal list of character boxes (and perhaps others) has `\hsize` as its default width, as do other boxes intended to be a line of type, i.e., the boxes made with the aid of the `\...line` macros (like `\leftline`, `\centerline`, and the like). Its height is the biggest of the heights of the boxes in its list, and the depth is the biggest depth of the boxes in it.

The box made up from a vertical list of boxes to give a page has default width `\hsize` and the sum of its height and depth is `\vsize`.

TeX can be asked explicitly to make up a box, by saying `\hbox{...}` if boxes are to be assembled horizontally, or saying `\vbox{...}` if boxes are to be assembled vertically, with the material `...` in braces providing the details. While being made, a box is only dimly aware of the context within which it is made.

When making up a box this way, you can explicitly specify its dimensions. For example, `\hbox to`⟨*dimen*⟩`{...}` will give you a box whose width is ⟨*dimen*⟩, while `\vbox to`⟨*dimen*⟩`{...}` will give you a box whose height and depth add up to ⟨*dimen*⟩. If the material `...` cannot be accomodated within that size, you'll get a complaint. If you are looking for a dimension that will just fit the given material plus the amount ⟨*dimen*⟩ extra, use `\hbox spread`⟨*dimen*⟩`{...}` (or `\vbox spread`⟨*dimen*⟩`{...}`) instead.

You can also let default determine the dimension of these made-up boxes. If the box was made from a horizontal list, then, much as with a line of type, its height is the biggest height among its boxes, and its depth is the biggest depth among its boxes, and its width is the sum of the widths of its boxes plus whatever glue was used in between. If the box was made from a vertical list, then the sum of its height and depth is the sum of the heights and depths of the boxes in its list, plus whatever interlineglue was used between these boxes. Further, its depth is that of its last (lowest) box, and its width is the width of its widest box. (If you prefer to have its height that of its first (topmost) box, don't ask for a `\vbox` but ask for a `\vtop{...}` instead.)

There are some perhaps surprising things implied by what I have described so far. Consider, for example, the command

\\`vbox`{`be`}

which asks for a box to be made from a vertical list. The description of the list is quite short, it's just `be`. In particular, the list begins with a character (box), hence TeX switches from the vertical mode you asked it to be in to making a line of type, in horizontal mode. Further, since this is the first line of type it is making for the present vertical list, it will start it with an indent (whose size is determined by `\parindent`). Then it will put in the `b` and then the `e`. With that, it reaches the end of the list, hence will finish off the line of type, making its width equal to `\hsize`, put the resulting box as the first and only box in the vertical list started by the command, and finish off the resulting box (whose width will also be `\hsize`; in fact, all its dimensions will be that of the line of type inside it). You can test just how wide the resulting box is by giving TeX the instruction `\hbox{\vbox{be}?}` as I am doing now, in a display:

be ?

The question mark is so far to the right, TeX complains about an overfull box, putting a black mark(!) into the right margin. Using a reasonable `\hsize` inside that `\vbox` will reduce that box's size, as in

be ?

As another example of how dimensions of a box are determined, consider the **rules**. A `\vrule` is a box (entirely filled with black) whose width is, by default, 0.4pt, and whose height and depth is, by default, that of the box in whose horizontal list it occurs. By contrast, an `\hrule` is a box (entirely filled with black) whose width is that of the box in whose vertical list it occurs while its height is .4pt and its depth is 0, by default. (In either case, it is possible to specify these dimensions arbitrarily, by saying things like `\hrule width`⟨*dimen*⟩ `height`⟨*dimen*⟩ etc.)

While we are on the subject of rules, they are often used with one of their dimension equal to 0. This means that you never see them printed. Such **empty rules** are useful nevertheless since they force the box within which they occur to have certain minimum size. For **example**, a `\mathstrut` is such a `\vrule` of 0 width, but of the same height and depth as a parenthesis. Hence any box made from a *horizontal* list containing such a `\mathstrut` will have height and depth at least that of a parenthesis. Such an empty `\hrule` is also useful at the beginning or the end of page, to give a `\vfill` something to push against (when trying to center material vertically on the page).

When making up a box from a list of boxes, the boxes are aligned (horizontally or vertically, depending on the mode) according to their reference points. You can change the alignment of an individual box by using `\raise`⟨*dimen*⟩ or `\lower`⟨*dimen*⟩ in front of it when in horizontal mode, and using `\moveright`⟨*dimen*⟩ or `\moveleft`⟨*dimen*⟩ when in vertical mode.

There are other box-making modes that TeX gets into, e.g., the math mode (single dollar signs) or the display mode (double dollar signs). Also, the vertical mode comes in two forms, the basic one when making up a page, and the internal vertical mode when acting on a `\vbox` command. The horizontal mode has two analogous versions.

In making up a page, lines are actually put together into paragraphs first, with the spacing between paragraphs controlled by a different parameter (viz. `\parskip`) than the spacing between lines (which is controlled by `\baselineskip`). Further, the first line of a paragraph is indented (according to the parameter `\parindent`).

If a line of text is meant to be a paragraph all by itself (such as a title or other displayed line), then it is good to so identify it by putting its material `...` into `\line{...}`. If the text in this line should be left-flush, use `\leftline{...}`, if right-flush, use `\rightline{...}`, and if centered, use `\centerline{...}`. The material `...` is expected to give a list of things suitable for a horizontal list.

TEST: If you were to submit the following two-line `.tex` file to TeX:

```
\hbox{this}\hbox{should}\vbox{be} \hbox{stacked} \ \hbox{this\ }\hbox{not}
\bye
```

what would the outcome be?

Did you predict the indents?

errors

Dealing with one's errors is perhaps the hardest and most frustrating part of TEX use.

One difficulty is TEX's very efficiency: There is little redundancy, hence *one* misplaced character can give a totally different (and often unexpected) interpretation to all subsequent instructions.

You defend yourself against this efficiency by checking out TEX instructions in small chunks. This means that you have a **working file**, in addition to the polished file that is to contain the finished set of instructions. The working file has the same formating material as the polished file. As you type your document, you would type a paragraph or two in the working file, submit that working file to TEX, and deal with the errors that TEX might point out to you. When the material is finally error-free, move it from the working file and add it to the material in the polished file.

You are bound to change that polished file later on (it is one of the attractions of using computer typesetting that it makes it so easy to change things!). Whenever you cannot really understand the resulting error(message)s, take the offending material from the polished file back into the working file and try to figure it out in that simpler environment.

In any case, TEX will give you error messages and expect some kind of response from you. Unfortunately, many of these error messages require a detailed understanding of the inner workings of TEX if they are to be understood fully. Fortunately, you can often deal with the problem without understanding the message fully.

For **example**, if TEX tells you that it is inserting a dollar sign or a brace, you can be sure that you need a dollar sign or a brace *somewhere*, and you can go looking for that place. Some editors (like **vi**) can help with that, as described below.

At times, it helps to ask for further explanations (by responding to the error message with an `h`). But often it requires an expert. If you are lucky, you know someone with that knowledge, and I'd exploit that situation. (You'll find that true TEXperts are often immensely proud of their hardwon knowledge and eager to show it off.)

If you and those around you don't understand an error message, simply tell TEX to continue, by hitting the RETURN or ENTER key.

On the other hand, there may be an error so confusing to TEX that it keeps complaining more or less about the same lines or things. Then you might as well give up and type an `e` to end the session. In any case, TEX keeps a running file of all these error messages (the file usually has the same name as your `.tex` file, but a different suffix, e.g., it might be a `.log` file or a `.lis` file). This means that you can look at these errors at leisure later on once TEX is finished with your file.

If you have a window system on your computer, you would have the `.tex` file in an editor in one window, and the `.log` file in another to look at while you try to take care of all the errors. Without a window system, you would jump between the `.tex` file and the `.log` file, reading in the `.log` file about the next error, then jumping to the `.tex` file to correct it.

You will know just where TEX has difficulty because any error message always refers to a line number in your `.tex` file. But, because of the aforementioned efficiency, this may only be the first place where an **earlier** error actually has some effect.

As an **example**, consider the very common error of forgetting to type the concluding dollar sign for some math expression. TEX will cruise along and interpret all subsequent instructions in math-mode terms until it comes to something that cannot be reasonably interpreted that way. It will then complain about *that* rather than about the dollar sign you forgot earlier. You can understand why it has to be that way: There is really no way of guessing just where that dollar sign should have been.

Because dollar signs go missing so often, TEX does add them at times just to avoid misinterpreting the entire remaining file. It will add a dollar sign if, while in math mode, it comes to the end of a grouping or of a paragraph that began before it entered the current math mode. And it will tell you about it.

If you have trouble finding the missing dollar sign, try to look for it systematically. Go to the beginning of your `.tex` file, set up your file editor to find the next dollar sign, then go systematically through the file from dollar sign to dollar sign, verifying that each intended beginning dollar sign has its appropriate ending dollar sign and vice versa. This procedure is not so hard when you only work on a little piece of the file at a time.

You might find that you are actually missing a *beginning* dollar sign. Those generate a different kind of error message since they are detected by TEX because something you intended to be in math-mode TEX is interpreting in text mode. There are all kinds of things (Greek letters and other math symbols, thin spaces like `\,`,`\;`, and math accents) which only make sense in math-mode. As soon as TEX comes across one of these in text mode, it puts in a dollar sign just prior to it, and tells you so. Unfortunately, this is often not exactly the right place. But you should be able to find the right place.

Even worse are the error messages due to **misplaced or missing braces**. Since TEX uses braces to indicate groupings, and all kinds of things depend on groupings, such mishandled braces can produce the most amazing error messages. TEX will put in a closing brace when it becomes clear that something is seriously wrong, just to close off that difficulty, and go on looking at the rest of the file. And it will tell you so.

The best defense against misplaced braces is to use an editor like **vi**. The `.exrc` file (which customizes **vi**) can be set up (by having in it the instruction `set sm`) so that, every time you type a closing brace, the editor will move the cursor temporarily to the matching opening brace. If it finds none, it will beep, and you are now alert to a mistake and in good position to do something about it. If it finds the match, you have a chance to see whether that is the match you intended and, again, you are in good position to do something about it. (There is a bonus here: once set up in this way, **vi** will also check for matching parentheses and brackets.)

The same facility gives you a chance for an **additional check** when you are done typing the file. Go to the end of the file and insert a closing brace. You expect a beep since all braces should already be matching; there should not be a opening brace matching this one. But, if the cursor moves to a brace, you now know that something is wrong about that (or some other) brace, and you can do something about it. To check for an **unmatched closing brace**, go to the beginning of the file and insert an opening brace. Of course, nothing will happen, since **vi** only checks when you type a closing fence. But, with the cursor on the brace you have just typed (and back in command mode), type `%`, and **vi** will look for the matching closing brace. You hope for the beep that says it can't find one. But if it does find one, there is another brace to do something about.

It is more efficient to look for misplaced braces in this way *before* submitting the file to TEX since you actually know the intentions of the instructions in your file, hence have a much easier time correcting misplaced braces and dollar signs than TEX does, as TEX is restricted to having to *divine* your intentions.

Missing or misplaced dollar signs cause TEX to switch between math and text mode in unintended ways. Not surprisingly, unintended switching between any two modes can be a source of seemingly strange errors (due to TEX's efficiency). (Mode switching, from ground to air or from air to ground, is also the most dangerous moment for airplanes.)

TEX is always in one of six possible modes. The primary four are: horizontal mode (when it puts together a line of text), vertical mode (when it puts together lines of text into paragraphs and pages), math-mode, and display mode. In addition, there are: restricted horizontal mode (when it assembles a list of boxes horizontally) and restricted vertical mode (when it assembles a list of boxes vertically).

Whenever TEX is in horizontal mode and it comes across an instruction incompatible with that (e.g., a `\par` or `\halign`), it will try to revert to a previous vertical mode by finishing the box it is presently assembling and adding it to the previously started vertical list. The trouble comes when there is no previously active vertical list. It will usually try to insert some brace (making a guess at the proper grouping), but you may have no idea what that is all about. But, once you diagnose the problem as one of unintended mode switching, i.e., an oversight on your part, then the remedy is simple. Start an encompassing vertical list by encasing the present context in a `\vbox{...}`. Whether in horizontal or vertical mode, TEX is always ready to start another box and, once that box is finished, it will simply add it to the list it was building when it came across the box-building instruction.

There is an analogous difficulty with unintended switching from vertical to horizontal mode, and, if due to an oversight, it is repaired by making the switch intentionally, i.e., by encasing the relevant material in an `\hbox{...}`.

TEXnicalities

Carl de Boor

Disclaimer and Exhortation

The statements about TEX made here are probably not completely true, but they are true enough to get you started. Knuth's TEXbook is the (seemingly inexhaustible) source for the complete information. My copy has several pages (e.g., 52, 135, 145, 147, 427, etc.) marked with paperclips. In fact, I have a copy of pages 434–438 handy to look up available standard symbols. You are urged to consult that book any time you feel ready. When you first do, follow the book's advice: Read it first from cover to cover **ignoring** ALL paragraphs marked with a dangerous bend sign. After that, you are ready for the more detailed stuff hidden in the rest of the book. You are also urged to look at the left pages of these notes to see how TEX was instructed to typeset what you read.

You might also want to secure a list of all the fonts available to TEX for your printer.

Appended to these notes is Sam Bent's TEX Reference Card which provides a superb ordered listing of the available commands in plain TEX.