

Stanford University Libraries  
Dept. of Special Collections

Col. UC 340 Title \_\_\_\_\_  
Box 7 Series 1986-052  
Fol. 50 Fol. Title TAN EPSILON CHI... NOV. 1978

Avron Barr

Stanford Artificial Intelligence Laboratory  
Memo AIM-317.1

November 1978  
(second printing)

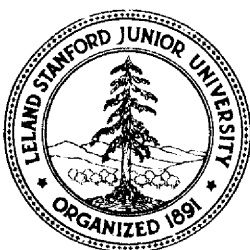
Computer Science Department  
Report No. STAN-CS-78-675.1

TAU EPSILON CHI  
A System for Technical Text

by

Donald E. Knuth

COMPUTER SCIENCE DEPARTMENT  
School of Humanities and Sciences  
STANFORD UNIVERSITY



Stanford Artificial Intelligence Laboratory  
Memo AIM-317.1

November 1978  
(second printing)

Computer Science Department  
Report No. STAN-CS-78-675

**TAU EPSILON CHI, a system for technical text**

© 1978 by D. E. Knuth

The author wishes to thank the many individuals who made helpful comments on the first drafts of this manual. Thanks are also due to the National Science Foundation, for helping to support the author's research under grant MCS 73-03752 A03, and to IBM Corporation for helping to defray publication expenses. Since the METAFONT system for typeface design is still under development, the type fonts used herein are only initial approximations to the eventual ones.

## TAU EPSILON CHI

### A SYSTEM FOR TECHNICAL TEXT

**G**ENTLE READER: This is a handbook about  $\TeX$ , a new typesetting system intended for the creation of beautiful books—and especially for books that contain a lot of mathematics. By preparing a manuscript in  $\TeX$  format, you will be telling a computer exactly how the manuscript is to be transformed into pages whose typographic quality is comparable to that of the world's finest printers; yet you won't need to do much more work than would be involved if you were simply typing the manuscript on an ordinary typewriter. In fact, your total work will probably be significantly less, if you consider the time it ordinarily takes to revise a typewritten manuscript, since computer text files are so easy to change and to reprocess. (If such claims sound too good to be true, keep in mind that they were made by  $\TeX$ 's designer, on a day when  $\TeX$  happened to be working, so the statements may be biased; but read on anyway.)

This manual is intended for people who have never used  $\TeX$  before, as well as for experienced  $\TeX$  hackers. In other words, it's the only manual there is. Everything you need to know about  $\TeX$  is explained here somewhere, and so are a lot of things that most users don't need to know. If you are preparing a simple manuscript, you won't need to know much about  $\TeX$  at all; on the other hand, some things that go into the printing of technical books are inherently difficult, and if you wish to achieve more complex effects you will want to penetrate into some of  $\TeX$ 's darker corners. In order to make it possible for many types of users



to read this manual effectively, a special symbol is used to designate material that is for wizards only: When the symbol



appears at the beginning of a paragraph, it warns of a "dangerous bend" in the train of thought; don't read the paragraph unless you need to. Brave and experienced drivers at the controls of  $\text{\TeX}$  will gradually enter more and more of these hazardous areas, but for most applications the details won't matter.

All that you really need to know before reading on is how to get a file of text into your computer using a standard editing program; this manual explains what that file ought to look like so that  $\text{\TeX}$  will understand it, but basic computer usage is not explained here. Some previous experience with technical typing will be quite helpful if you plan to do heavily mathematical work with  $\text{\TeX}$ , although it is not absolutely necessary.  $\text{\TeX}$  will do most of the necessary formatting of equations automatically; but users with more experience will be able to obtain better results, since there are so many ways to deal with formulas.

Computer system manuals usually make dull reading, but take heart: This one contains *JOKES* every once in a while, so you might actually enjoy reading it. (However, most of the jokes can only be appreciated properly if you understand a technical point that is being made—so read *carefully*.)

Another somewhat unique characteristic of this manual is that it doesn't always tell the truth. When informally introducing certain  $\text{\TeX}$  concepts, general rules will be stated, but later you will find that they aren't strictly true. The author feels that this technique of deliberate lying will actually make it easier for you to learn the concepts; once you learn a simple but false rule, it will not be hard to supplement that rule with its exceptions.

In order to help you internalize what you're reading, occasional *EXERCISES* are sprinkled through this manual. It is generally intended that every reader should try every exercise, except those exercises which appear in the "dangerous bend" areas. If you can't solve the problem, you can always look at the answers that appear at the end of the manual. But please, try first to solve it by yourself; then you'll learn more and you'll learn faster. Furthermore, if you think you do know the answer to an exercise, you should turn to the answer pages (Appendix A) and check it out just to make sure.

**CONTENTS**

1. The name of the game _____	4
2. Book printing versus ordinary typing _____	4
3. Controlling $\TeX$ _____	7
4. Fonts of type _____	12
5. Grouping _____	15
6. Running $\TeX$ _____	18
7. How $\TeX$ reads what you type _____	28
8. The characters you type _____	33
9. $\TeX$ 's standard roman fonts _____	36
10. Dimensions _____	40
11. Boxes _____	41
12. Glue _____	45
13. Modes _____	50
14. How $\TeX$ breaks paragraphs into lines _____	52
15. How $\TeX$ makes lists of lines into pages _____	57
16. Typing math formulas _____	60
17. More about math _____	64
18. Fine points of mathematics typing _____	71
19. Displayed equations _____	91
20. Definitions (also called macros) _____	96
21. Making boxes _____	99
22. Alignment _____	104
23. Output routines _____	109
24. Summary of vertical mode _____	114
25. Summary of horizontal mode _____	121
26. Summary of math mode _____	130
27. Recovery from errors _____	138
A. Answers to all the exercises _____	148
B. Basic $\TeX$ format _____	151
E. Example of a book format _____	154
F. Font tables _____	168
H. Hyphenation _____	180
I. Index _____	187
S. Special notes about using $\TeX$ at Stanford _____	198

### <1> The name of the game

English words like "technology" stem from a Greek root beginning with the letters  $\tau\epsilon\chi\dots$ ; and this same Greek word means art as well as technology. Hence the name  $\text{\TeX}$ , which is an upper-case form of  $\tau\epsilon\chi$ .

Insiders pronounce the  $\chi$  of  $\text{\TeX}$  as a Greek chi, not as an "x", so that  $\text{\TeX}$  rhymes with the word blecchhh. It's the "ch" sound in Scottish words like *loch* or German words like *ach*; it's a Spanish "j" and a Russian "kh". When you say it properly to your computer, the terminal may become slightly moist.

The purpose of this pronunciation exercise is to remind you that  $\text{\TeX}$  is primarily concerned with high-quality technical manuscripts: its emphasis is on art and technology, as in the underlying Greek word. If you merely want to produce passably good quality—something acceptable and basically readable but not really beautiful—a simpler system will usually suffice. With  $\text{\TeX}$  the goal is to produce the *finest* quality; this requires more attention to detail, but fortunately it is not that much harder to go this extra distance, and you can take special pride in the finished product.

On the other hand you might find it more comfortable to pronounce  $\text{\TeX}$  as a Texan would and to shrug off all this high-falutin' nonsense about beauty and quality. Go ahead and do what you want, the computer won't mind.

### <2> Book printing versus ordinary typing

When you first started using a computer terminal, you probably had to adjust to the difference between the digit "1" and the lower case letter "l". When you take the next step to the level of typography that is common in book publishing, a few more adjustments of the same kind need to be made.

In the first place, there are two kinds of quotation marks in books, but only one kind on the typewriter. Even on your computer terminal, which has more characters than an ordinary typewriter, you probably have only a non-oriented double-quote mark ("), because the standard "ascii" code for computers was not invented with book publishing in mind. However, your terminal probably does have two flavors of single-quote marks, namely ' and ', which you can get by typing ` and ´. The second of these is useful also as an apostrophe.

To produce double-quote marks with  $\text{\TeX}$ , you simply type two single-quote

marks of the appropriate kind. For example, to produce an output like

"I understand."

(including the quotation marks) you would type

`"I understand."`

on your terminal.

A typewriter-like style of type will be used throughout this manual to indicate  $\text{\TeX}$  constructions you might type on your terminal, so that the symbols actually typed are readily distinguishable from the output  $\text{\TeX}$  would produce and from the comments in the manual itself. Here are the symbols to be used in the examples:

```

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789"#$%&@*+==,.;?!
()<>{}[]`^~'±↓←\|/≠∞

```

If these are not all on your computer terminal, do not despair;  $\text{\TeX}$  can make do with the ones you have. One additional symbol

□

is also used to stand for a *blank space*, in case it is important to emphasize that a blank space is typed; without such a symbol you would have difficulty seeing the invisible parts of certain examples.

Another important distinction between book printing and ordinary typing is the use of dashes, hyphens, and minus signs. In good math books, these symbols are all different; in fact there are usually at least four different symbols in use:

- a hyphen (-);
- an en-dash (–);
- an em-dash (—);
- a minus sign (−).

Hyphens are used for compound words like "nitty-gritty" and "Fawcett-Majors". En-dashes are used for number ranges like "pages 13–34" and also in contexts


like “exercise 1.2.6–52”. Em-dashes are used for punctuation in sentences—they are what we often call simply dashes. And minus signs are used in formulas. A conscientious user of  $\TeX$  will be careful to distinguish these four usages, and here is how to do it:

- for a hyphen, type a hyphen (-);
- for an en-dash, type two hyphens (--);
- for an em-dash, type three hyphens (---);
- for a minus sign, type a hyphen in mathematics mode ( $\$-\$$ ).

(Mathematics mode occurs between dollar signs; it is discussed later, so you needn't worry about it now.)

If you look closely at most well-printed books, you will find that certain combinations of letters are treated as a unit. For example, this is true of the “f” and the “i” of “find”. Such combinations are called *ligatures*, and professional typesetters have traditionally been trained to watch for letter pairs such as *ff*, *fi*, *fl*, *ffi*, and *ffl*. (It's somewhat surprising how often these combinations appear.) Fortunately you do *not* have to concern yourself with ligatures, since  $\TeX$  is perfectly capable of handling such things by itself. In fact,  $\TeX$  will also look for combinations of adjacent letters (like “A” next to “V”) that ought to be moved closer together for better appearance; this is called  *Kerning*.

To summarize this chapter: When using  $\TeX$  for straight copy, you type the copy as on an ordinary typewriter, except that you need to be careful about quotation marks, the number 1, and various kinds of hyphens/dashes.  $\TeX$  will take care of other niceties like ligatures and kerning.

 In case you need to type quotes within quotes, for example a single quote followed by a double quote, you can't simply type `' '` because  $\TeX$  will interpret this as `"'` (namely, double-quote followed by single-quote). If you have already read Chapter 5, you might expect that the solution will be to use grouping—namely, to type something like `{'}```. But it turns out that this doesn't produce the desired result, because there is usually more space following a double quote than there is following a single quote: What you get is `"'`, which is indeed a single quote followed by a double quote (if you look at it closely enough), but it looks almost like three equally-spaced single quotes. On the other hand, you certainly won't want to type `{}'``, because this space is much too large—just as large as the space between words—and  $\TeX$  might even start a new line at such a space when making up a paragraph! There are at least two ways to solve`

the problem, both of which involve more complicated features of T<sub>E</sub>X that we shall study later. First, if you have a definition such as

```
\def\2{\hjust to 2pt{}}
```

in the format of your manuscript, you can type `“\2”`. This definition puts 2 points of blank space between the quotes, so the result is `”`; you could, of course, vary the amount of space, or define another control sequence besides `\2` for this purpose. Second, you could use the idea of “thin space” in math formulas: namely, if you type `“$\",$”` the result will be `”`.



►Exercise 2.1: OK, now you know how to produce `”` and `”`; how do you get `“` and `“`?

### <3> Controlling T<sub>E</sub>X

Your keyboard has very few keys compared to the large number of symbols you may want to specify. In order to make a limited keyboard sufficiently versatile, one of the characters you can type is reserved for special use, and it is called the *escape character*. Whenever you want to type something that controls the format of your manuscript, or something that doesn't use the keyboard in the ordinary way, you type the escape character followed by an indication of what you want to do.

You get to choose your own escape character. It can be any typeable symbol, preferably some character found in a reasonably convenient location on your keyboard, yet it should be a symbol that is rarely (if ever) used in the manuscript you are typing. For our purposes in this manual, the “backslash” character `“\”` will be used as the escape in all the examples. You may wish to adopt backslash as your personal escape symbol, but T<sub>E</sub>X doesn't have any character built in for this purpose. In fact, T<sub>E</sub>X always takes the *first nonblank character* you give it and assumes that it is to be your escape character.

Note: Some computer terminals have a key marked “ESC”, but that is *not* your escape character! It is a key that sends a special message to the operating system, so don't confuse it with what this manual calls “escape”.

Immediately after typing `“\”` (i.e., immediately after an escape character) you type a coded command telling T<sub>E</sub>X what you have in mind. Such commands

are called *control sequences*. For example, you might type

```
\input ms
```

which (as we will see later) causes  $\text{\TeX}$  to begin reading a file called "ms.TEX"; the string of characters "\input" is a control sequence. Here's another example:

```
George P\`olya and Gabor Szeg\"o.
```

$\text{\TeX}$  converts this to "George Pólya and Gabor Szegő." There are two control sequences, \` and \", in this example, and they are used to indicate the special accents.

Control sequences come in two flavors. The first kind, like \input, consists of the escape character followed by one or more letters, followed by a space or by something besides a letter. ( $\text{\TeX}$  has to know where the control sequence ends, so you have to put a space after a control sequence if the following character is a letter; for example, if you type "\inputms",  $\text{\TeX}$  will interpret this as a control sequence with seven letters.) The second variety of control sequence, like \`, consists of the escape character followed by a single *nonletter*. In this case you don't need a space to separate the control sequence from a letter that follows, since control sequences of the second kind always have a single symbol after the escape.

When a space comes after a control sequence (of either kind), it is ignored by  $\text{\TeX}$ ; i.e., it is not considered to be a "real" space belonging to the manuscript being typeset. Thus, the example above could have been typed as

```
George P\` olya and Gabor Szeg\" o.
```

$\text{\TeX}$  will treat both examples the same way; it always discards spaces after control sequences.

So the question arises, what do you do if you actually want a space to appear after a control sequence? We will see later that  $\text{\TeX}$  treats two or more consecutive spaces as a single space, so the answer is *not* going to be "type two spaces." The correct answer is to type "escape space", namely

```
\U
```

(the escape character followed by a blank space); TeX will treat this as a space not to be ignored. Note that escape-space is a control sequence of the second kind, since there is a single nonletter (`\`) following the escape character. According to the rules, further spaces immediately following `\` will be ignored, but if you want to enter, say, three consecutive spaces into a manuscript you can type `"\ \ \ "`. Incidentally, typists are often taught to put two spaces at the ends of sentences; but we will see later that TeX has its own way to produce extra space in such cases. Thus you needn't be consistent in the number of spaces you type.

It is usually unnecessary for you to use "escape space", since control sequences aren't often needed at the ends of words. But here's an example that might shed some light on the matter: This manual itself has been typeset by TeX, and one of the things that occurs fairly often is the tricky logo "TeX", which requires backspacing and lowering the E. We will see below that it is possible for any user to define new control sequences to stand as abbreviations of commonly occurring constructions; and at the beginning of this manual, a special definition was made so that the control sequence

```
\TeX
```

would produce the instructions necessary to typeset "TeX". When a phrase like "TeX ignores spaces after control sequences." is to be typeset, the manuscript renders it as follows:

```
\TeX\ ignores spaces after control sequences.
```

Notice the extra `\` following `\TeX`; this produces the escape-space that is necessary because TeX ignores spaces after control sequences. Without this extra `\`, the result would have been

```
TeXignores spaces after control sequences.
```

Consider also what happens if `\TeX` is not followed by a space, as in

```
the logo ``\TeX''.
```

It would be permissible to put a blank space after the X, but not an escape character; if the manuscript were changed to read

```
the logo ``\TeX\''
```



the result would be curious indeed—can you guess it? Answer: The `\`` would be a control sequence denoting an acute accent, as in our `P\`olya` example above; the effect would therefore be to put an accent over the next nonblank character, which as it happens is a single-quote mark. In other words, the result would be

the logo "T $\acute{E}$ X"

because the ligature that changes ``` into `"` is not recognized.

►**Exercise 3.1:** State two ways to specify the French word "mathématique". Can you guess how the word "centimètre" should be specified?

$\TeX$  understands almost 300 control sequences as part of its standard built-in vocabulary, and all of these are explained in this manual somewhere. Fortunately you won't have too much trouble learning them, since the vast majority are simply the names of special characters used in mathematical formulas. For example, the control sequences `\Ascr`, `\Bscr`, ..., `\Zscr` stand for the upper case script letters  $\mathcal{A}$ ,  $\mathcal{B}$ , ...,  $\mathcal{Z}$ ; and you can type `\aleph` to get  $\aleph$ , `\doteq` to get  $\doteq$ , `\oplus` to get  $\oplus$ , `\leftarrow` to get  $\leftarrow$ , etc.

As mentioned above,  $\TeX$  can be taught to understand other control sequences besides those in its primitive vocabulary. For example, `\TEX` is not one of the standard control sequences; it had to be defined specially for producing this manual. In general there will be special control sequences that define the *style* of a book or a series of books: they will be used at the beginning of chapters, or to handle special formats such as might be used in a bibliography, etc. Such style-defining control sequences are usually defined once and for all by  $\TeX$  experts skilled in the lore of control-sequence definition, and novice  $\TeX$  users don't have to worry about the job of defining any new control sequences; the only problem is to learn how to use somebody else's definitions. (The person who designs a  $\TeX$  style is obliged to write a supplement to this manual explaining how to use his or her control sequences.)

In this manual we shall frequently refer to a so-called "basic  $\TeX$  style" consisting of the definitions in Appendix B, since these basic definitions have proved to be useful for common one-shot jobs; and since they probably also will be included as a part of more elaborate styles. Appendix E contains an example of a more elaborate style, namely the definitions used to typeset D. E. Knuth's series of books on *The Art of Computer Programming*. There's no need for you to look at these appendices now, they are included only for reference purposes.

The main point of these remarks, as far as novice T<sub>E</sub>X users are concerned, is that it is indeed possible to define nonstandard T<sub>E</sub>X control sequences, but it can be tricky. You can safely rely on the standard control sequences, and on the basic extensions defined in Appendix B (which will be explained later in this manual), until you become an experienced T<sub>E</sub>Xnical typist.

⊠ Those of you who wish to define control sequences should know that T<sub>E</sub>X has further rules about them, namely that many different spellings of the same control sequence may be possible. This fact allows T<sub>E</sub>X to handle control sequences quite efficiently; and T<sub>E</sub>X's usefulness is not seriously affected, because new control sequences aren't needed very often. A control sequence of the first kind (i.e., one consisting of letters only) may involve both upper case and lower case letters, but the distinction between cases is ignored after the first letter. Thus `\TeX` could also be typed "`\TEX`" or "`\Tex`" or "`\TEx`"—each of these four has the same meaning and the same effect. But "`\tex`" would *not* be the same, because there is a case distinction on the first letter. (Typing "`\gamma`" results in  $\gamma$ , but "`\Gamma`" or "`\GAMMA`" results in  $\Gamma$ .)

⊠ Another rule takes over when there are seven or more letters after the escape: all letters after the seventh are replaced by "x", and then groups of eight letters are removed if necessary until at most 14 letters are left. Thus `\underline` is the same as `\underl1xx`; and it is also the same as `\underlinedsymbols` or any other control sequence that starts with `\u` followed by n or N, then d or D, then e or E, then r or R, then l or L, then i or I, then 2 or 10 or 18 or 26 or ... letters. But `\underline` is not the same as `\underlines`, because these two control sequences don't have the same length modulo 8.

⊠ As a consequence of these rules, there are 128 essentially distinct control sequences of length two—namely, escape followed by any 7-bit character, whether a letter or not. There are  $52 \times 26$  essentially distinct control sequences of length three, because there are  $26 + 26 = 52$  choices for the first letter following the escape and 26 different choices for the second letter; there are  $52 \times 26 \times 26$  essentially distinct control sequences of length four,  $52 \times 26 \times 26 \times 26$  of length five,  $52 \times 26 \times 26 \times 26 \times 26$  of length six,  $52 \times 26 \times 26 \times 26 \times 26 \times 26$  of length seven. There are  $52 \times 26 \times 26 \times 26 \times 26 \times 26 \times 26$  essentially distinct control sequences of length 8 plus a multiple of 8, and the same number holds for length 9 plus a multiple of 8, ..., length 15 plus a multiple of 8. Thus the total number of distinct control sequences available is exactly

$$128 + 52 \cdot 26 + 52 \cdot 26^2 + 52 \cdot 26^3 + 52 \cdot 26^4 + 52 \cdot 26^5 + 8 \cdot 52 \cdot 26^6 = 129151507704;$$

that should be enough. Even though T<sub>E</sub>X accepts alternative spellings, you should be consistent in each manuscript, since some implementations of T<sub>E</sub>X may not be exactly the same in this respect.

⊠ Nonprinting control characters like `\carriage-return` might follow an escape character, and these lead to distinct control sequences according to the rules. Initially TeX is set up to treat `\tab` and `\line-feed` and `\vertical-tab` and `\form-feed` and `\carriage-return` the same as `\`  (escape space); it is recommended that none of these six control sequences be redefined.

#### <4> Fonts of type

Occasionally you will want to change from one typeface to another, for example if you wish to be **bold** or to *emphasize* something. TeX deals with sets of 128 characters called "fonts" of type, and the control sequence `\:` is used to select a particular font. If, for example, fonts `n`, `b`, and `s` have been predefined to represent normal, bold, and slanted styles of type, you might specify the last few words of the first sentence of this paragraph in the following way:

```
to be \:b bold \:n or to \:s emphasize \:n something.
```

(Blank spaces after font codes like `b` are ignored by TeX just like the spaces after control sequences; furthermore, since a font code is always of length 1, you don't need a space after it. Thus, `\:bbold` would be treated the same as `\:bLLbold`. It is probably best to type a space after the font codes, even though you don't really need one, for the sake of readability.)

You probably will never\* use the `\:` sequence yourself, since the predesigned format you are using usually includes special control sequences that give symbolic names to the fonts. For example, the "basic TeX format" in Appendix B defines three control sequences for this purpose.

```
\rm switches to the normal "Roman" typeface: Roman
\sl switches to a slanted typeface:           Slanted
\bf switches to a boldface style:             Bold
```

With such a system, you can type the above example as

```
to be \bf bold \rm or to \sl emphasize \rm something.
```

---

\*Well..., hardly ever.

The advantage of such control sequences is that you can use the same abbreviations `\rm`, `\sl`, `\bf` in any size of type, although different font codes are actually used for different sizes. For example, fonts `a`, `n`, `q` might be the normal, slanted, and bold fonts in a standard "10-point" size of type, while `c`, `p`, `s` might be the corresponding fonts in a smaller "8-point" size. It would be difficult to remember how the codes change in different sizes. So the *Art of Computer Programming* book design in Appendix E allows you to say

`\tenpoint`

whenever you are beginning to type material that belongs in 10-point size, after which `\rm` will be equivalent to `\:a`, and `\sl` will be equivalent to `\:n`, etc. Now if you switch to 8-point size (in a footnote, say) the instruction


`\eightpoint`

(which appears in the `\footnote` format) will cause `\sl` to be equivalent to `\:p`. All you need to remember is the abbreviations `\rm`, `\sl`, and `\bf` regardless of what type size you are using.

There actually is a better way yet to handle the above example, using `TEX`'s "grouping" feature, which we shall discuss in the next chapter. With this feature you would type

to be `{\bf bold}` or to `{\sl emphasize}` something.

As we will see, switching fonts within `{` and `}` does not affect the fonts outside, so you don't need to say explicitly that you are returning to `\rm` in this scheme. Thus, you can pretty much forget about the other ways we have been discussing for font switching; it's best to use grouping.

 When you do use the `\:` instruction to change fonts, here are the rules you need to know. `TEX` can handle up to 32 different fonts in any particular job (counting different sizes of the same style). These 32 fonts are distinguished by the least significant five bits of the 7-bit ascii character code you type following `\:`; if you don't understand

what this means, use the following code names for your fonts:

Internal font number	TEX font code	Internal font number	TEX font code	Internal font number	TEX font code	Internal font number	TEX font code
1	⊙ or ˘	9	H or h	17	P or p	25	X or x
2	A or a	10	I or i	18	Q or q	26	Y or y
3	B or b	11	J or j	19	R or r	27	Z or z
4	C or c	12	K or k	20	S or s	28	[ or ;
5	D or d	13	L or l	21	T or t	29	< or ≤
6	E or e	14	M or m	22	U or u	30	] or =
7	F or f	15	N or n	23	V or v	31	> or †
8	G or g	16	O or o	24	W or w	32	? or †

You never refer to a font by its number, always by its code. Code A is treated the same as a, etc.; but a wise typist will consistently use the same codes in any particular manuscript, because later TEXes may allow more than 32 fonts.

⚡ Of course TEX can make use of hundreds of different fonts in different jobs. The 32-font restriction applies only within a particular job, because TEX doesn't want to keep the details about more than  $32 \times 128 = 4096$  characters in its memory at once; there isn't enough room. Thus the internal font codes will refer, in general, to different "real" fonts. The first time you use a font code, you must *define* it by giving the full name of the font in the system's collection. For example, when the basic TEX format in Appendix B says

```
\:a=cmr10
```

this selects font code a and defines it to be the system's font "cmr10", an abbreviation for "Computer Modern Roman 10 point". The rule for defining a font is that the font code (a in this example) must be followed immediately by "=" or "+" (not a space) when it first appears, and this must be followed immediately by the system name of the font file; then comes a blank space to denote the end of the font file name.

⚡ Once a font code is defined, it can never be redefined again. Thus if you type, say, "\:a=cmr10" when font code a has already been defined, the characters "=cmr10" will be treated as part of your manuscript, and they will dutifully be set into type (in font a). It's best to define all your fonts in format specifications at the very beginning of your input.

When you change fonts within a line, TEX will line the letters up according to their "baselines." For example, suppose that font codes a, b, c, d, e, f refer

respectively to 10-point, 9-point, 8-point, 7-point, 6-point, and 5-point roman fonts; then if you type

```
\:a smaller \:b and smaller \:c and smaller
\:d and smaller \:e and smaller \:f and smaller \:a
```

the result is smaller and smaller and smaller and smaller and smaller and smaller. Of course this is something authors don't do very often at the moment, because printers can't do such things easily with traditional lead types. Perhaps poets who wish to speak in a still small voice will cause future books to make use of frequent font variations, but nowadays it's only an occasional font freak (like the author of this manual) who likes it. One should not get too carried away by the prospect of font switching unless there is good reason.

►**Exercise 4.1:** Explain how to type the bibliographic reference "Ulrich Dieter, *Journal für die reine und angewandte Mathematik* 201 (1959), 37–70."

## <5> Grouping

Every once in a while it is necessary to treat part of a manuscript as a unit, so you need to indicate in some fashion where that part begins and ends. For this purpose TeX gives special interpretation to two "grouping characters" (just as it treats the escape character in a special way). We shall assume in this manual that { and } are the grouping characters, although any other typeable characters may be reserved for this function.

We saw one example of grouping in the previous chapter, where it was pointed out that font changes inside a group do not affect the fonts in force outside. This gives the effect of what computer scientists call "block structure." Another example of grouping occurs when you are using certain control sequences; for example, if you want to center something on a line you can type

```
\ctrline{This information will be centered.}
```

using the control sequence `\ctrline` defined in basic TeX format (Appendix B).

Grouping is used in quite a few of TeX's more complex instructions, although it is largely unnecessary in simple manuscripts. Here's an example of a slightly more complex case, the definition of a new control sequence `\rm` as mentioned

in the previous chapter:

```
\def\rm{\:a}
```

This means that control sequence `\rm` is henceforth to be replaced in the input by the control sequence `\:` followed by `a`. One can also have *groups within groups*, e.g.,

```
\def\tenpoint{\def\rm{\:a}\def\s1{\:n}\def\bf{\:q}}
```

which means that the control sequence `\tenpoint` is henceforth to be replaced in the input by

```
\def\rm{\:a}\def\s1{\:n}\def\bf{\:q}
```

and these, in turn, describe replacements for the control sequences `\rm`, `\s1`, and `\bf`. If you are a novice `TEX` user, you will probably not be using `\def` yourself to define control sequences; the point of this example is merely to demonstrate that groups can indeed arise within groups.

⚠ Groups within groups will happen only in rather complicated situations, but in such cases it is extremely important that you don't leave out a `{` or a `}`, lest `TEX` get hopelessly confused. For example, the `\output` routine in Appendix E has as many as five levels of groups within groups within ...; although each level is fairly simple by itself, the total cumulative effect can boggle the mind, so the author had to try three times before getting the `{`'s and `}`'s right. In such situations there is a handy rule for figuring out which `{` goes with which `}`, and whether or not you have forgotten any braces. Start with a mental count of zero, and go from left to right in your `TEX` input. When you get to a `{`, add one to the count, and write the resulting number lightly above the `{`. When you get to a `}`, write the current count lightly above it and then subtract one from the count. For example,

```

          1  2  2  2  3  3  3  3  2  2  2  1
          ...{...{...}...{...{...}...{...}...}...{...}...}...
Current count: 0  1  2  1  2  3  2  3  2  1  2  1  0

```

If the input is properly grouped, your count will return to zero, and it will never become less than zero. The `{` corresponding to any particular `}` is the nearest preceding `{` having the same number as the `}`. (You need not apply this procedure to the entire input manuscript, just to any part that is supposed to be understood as a unit. For example, you can apply this procedure to the right-hand side of any definition that uses `\def`.)

Suppose that you had typed

```
\ctrline{This information will be {\sl centered}.}
```

Then you would have gotten

This information will be centered.

Now suppose that you type

```
\ctrline{This information will be {centered}.}
```

What do you think will happen? Answer: you will get

This information will be centered.

The result looks just as if those innermost braces had not appeared at all, because you haven't used the grouping to change fonts or anything.  $\TeX$  doesn't mind if you want to waste your time making groups for no reason.

Actually there is a reason why you might want to use grouping without font changes, etc., namely when you want to make sure that spacing comes out right. In Chapter 3 we discussed the control sequence  $\backslash\text{TEX}$  that the author of this manual has used to get the logo " $\text{\TeX}$ ", and we observed that the space after  $\backslash\text{TEX}$  is ignored since  $\backslash\text{TEX}$  is a control sequence. Thus it was apparently necessary to type " $\backslash\text{TEX}\backslash$ " when there was supposed to be a space following " $\text{\TeX}$ ", but it was a mistake to type " $\backslash\text{TEX}\backslash$ " when the next character was to be a punctuation mark or something else besides a space. Well, in *all* cases it would be correct to type

$$\{\backslash\text{TEX}\}$$

whether or not the following character is a space, because the  $\}$  stops  $\text{\TeX}$  from looking for the optional space after  $\backslash\text{TEX}$ . This might come in handy when you're using a text editor (e.g., when replacing all occurrences of a particular word by a control sequence). Another thing you could do is type

$$\backslash\text{TEX}\{\}$$

using an empty group for the same purpose: the  $\{\}$  here is a group of no characters, so it produces no output, but it does have the effect of shutting off  $\text{\TeX}$ 's scan for blanks.

►**Exercise 5.1:** Suppose you want to specify two hyphens in a row; you can't type "--" because  $\text{\TeX}$  will read that as an en-dash, so what can you do?



⊠ When  $\TeX$  starts any job, all characters are alike; there is no escape character, and there are no grouping characters.  $\TeX$  automatically makes the first nonblank input character the escape, but if a manuscript is going to use grouping, the grouping characters must be “turned on.” The basic format in Appendix B does this, and you can do it yourself in the following way: Type “ $\backslash\text{chcode}\langle\text{number}\rangle+1$ ” for the left delimiter and “ $\backslash\text{chcode}\langle\text{number}\rangle+2$ ” for the right delimiter, where  $\langle\text{number}\rangle$  is the numeric value of the 7-bit code for the desired character. For example, “{” and “}” have the respective codes ‘173 and ‘176 at Stanford—this is a local deviation from some ascii codes at other places—so the instructions

```
 $\backslash\text{chcode}'173+1 \backslash\text{chcode}'176+2$ 
```

appear among the basic format definitions in Appendix B. (Numbers beginning with ‘ are in octal notation, cf. Chapter 8.) It is possible to have several characters simultaneously serving as group delimiters, simply by using  $\backslash\text{chcode}$  to specify each of them.

⊠ Font changes are not the only things that “stay inside” a group without affecting the text outside. This same localization applies to any control sequences defined within the group (except those using  $\backslash\text{gdef}$  in place of  $\backslash\text{def}$ ); to glue-spacing parameters such as those set by  $\backslash\text{baselineskip}$  and  $\backslash\text{tabskip}$ ; to  $\TeX$  control parameters such as those set by  $\backslash\text{trace}$  and  $\backslash\text{jpar}$ ; and to the character interpretations set by  $\backslash\text{chcode}$ . But localization does *not* apply to definitions of  $\backslash\text{output}$  routines, or to the size parameters set by  $\backslash\text{hsize}$ ,  $\backslash\text{vsize}$ ,  $\backslash\text{parindent}$ ,  $\backslash\text{maxdepth}$ , and  $\backslash\text{topbaseline}$ . Furthermore, if you type “ $\{\backslash:a=\text{cmr10}\}$ ”, the “cmr10” part of this font definition still is irrevocably tied to code a.

⊠ ▶Exercise 5.2: Would  $\backslash\text{def}\backslash\text{rm}\{\{\backslash:a\}\}$  have the same effect as the definition  $\backslash\text{def}\backslash\text{rm}\{\backslash:a\}$ ? (The only difference is an extra level of grouping.)

⊠ ▶Exercise 5.3: Suppose  $\backslash\text{chcode}'74+1 \backslash\text{chcode}'76+2$  appears near the beginning of a group that begins with {; these specifications instruct  $\TeX$  to treat < and > as group delimiters. According to the rules above, the characters < and > will revert to their previous meaning when the group ends; but should the group end with } or with >?

## <6> Running $\TeX$

The best way to learn how to do something is to do it, and the best way to learn how to use  $\TeX$  is to use it. Thus, it's high time for you to sit down at a computer terminal and interact with the  $\TeX$  system, trying things out to see

what happens. Here are some small but complete examples suggested for your first encounter. The examples are presented in terms of the Stanford WAITS system; slightly different conventions may be in use at other installations.

Caution: This chapter is rather a long one. Why don't you stop reading now, and come back to this tomorrow?

OK, let's suppose that you're rested and excited about having a trial run of T<sub>E</sub>X. Step-by-step instructions for using it appear in this chapter. First do this: Go to the lab where the graphic output device is, since you will be wanting to see the output that you get—it won't really be satisfactory to generate new copy with T<sub>E</sub>X from a remote location. Then log in; and when the operating system types "." at you, type back

```
r tex
```

(followed by (carriage-return)). This causes T<sub>E</sub>X to start up, and when it is ready it will type "\*". Now type

```
\input basic
```

and (carriage-return); this causes the basic T<sub>E</sub>X format of Appendix B to be read into the system. T<sub>E</sub>X will type

```
(basic.TEX 1 2 3 4)
```

on your terminal as it is processing this material, meaning that it has read pages 1, 2, 3, and 4 of this file. Then it types "\*", waiting for more input. At this point the \rm font has been selected, which is the "normal" cmr10 font, and T<sub>E</sub>X is ready to accept an input manuscript using the basic conventions.

Now type several more lines, each followed by (carriage-return):

```
\hsize 2 in  
\vskip 1 in  
\ctrline{MY STORY}  
\vskip 36 pt  
\ctrline{\sl by A. U. Thor}  
\vskip 2.54 cm
```

```

Once upon a time, in a distant
galaxy called \error \"0\"o\c c,
there lived a computer
named R. J. Drofnats. \par
Mr. Drofnats---or ``R. J.,'' as
he preferred to be called---
was lousy at typesetting, but he
had other nice qualities. For
example, he gave error messages
when a typist forgot to end a paragraph
properly. \end
\par\vfill\end

```

This example is a bit long, and more than a bit silly, but it's no trick for a good typist like you and it will give you some worthwhile experience, so please do it. For your own good.

Incidentally, the example introduces a few more features that you might as well learn as you are typing, so it's probably best for you to type a line, then read the explanation that appears below, then type the next line and so on.

The instruction "`\hsize 2 in`" says that rather narrow lines will be set, only 2 inches wide. (On a low-resolution device like the XGP currently used at Stanford, "2 in" really means about 2.6 inches, because  $\TeX$  expects that its output on such devices will be used only for proofreading, or that the output will be reduced to about 77% of its physical size before actual printing. The 10-point type `cmr10` will actually appear to be essentially the same size as 13-point type in books; in other words, you should expect to see output "larger than life.")

The instruction "`\vskip 1 in`" means a *vertical skip* of one inch. (Really 1.3 inches, on an XGP or VERSATEC, but from now on we won't mention this expansion.) Then the instruction "`\ctrline{MY STORY}`" causes a line of type that says "MY STORY" to be centered in the 2-inch column. (Recall from Chapter 5 that  $\TeX$ 's basic formats, which we loaded by typing "`\input basic`", include this `\ctrline` and grouping facility for centering things.)

The instruction "`\vskip 36 pt`" is another vertical skip, this time by the amount 36 points—which is a printer's measure slightly less than half an inch. Book measurements have traditionally been specified in units of picas and points, and  $\TeX$  does not want to shake printers up too badly, so it allows a variety of

different units of length to be specified.

The instruction "`\ctrline{\s1 by A. U. Thor}`" makes another centered line, this time in the slanted 10-point font (because of the `\s1`). This `\s1` is inside a group, so it doesn't affect the type style being used elsewhere.

You can probably guess what "`\vskip 2.54 cm`" means; or aren't you ready for the metric system yet? It turns out that 2.54 centimeters is exactly one inch.

The next line begins the straight text, which is what you will be typing most of the time; don't be dismayed by the messy spacing instructions like `\vskip` that you have been typing so far. Something messy like that is expected at the beginning of a manuscript, but it doesn't last long. When T<sub>E</sub>X begins to read the words

Once upon a time, in a distant

it starts up a new paragraph. Now comes the good news, if you haven't used computer typesetting before: You don't have to worry about where to break lines in the paragraph, T<sub>E</sub>X will do that for you. You can type long lines or short lines, it doesn't matter; *every time you hit (carriage-return) it is essentially the same as typing a space*. When T<sub>E</sub>X has read the entire paragraph, it will try to break up the text so that each line of output, except the last, contains about the same amount of copy; and it will hyphenate words if necessary (but only as a last resort).

After you type in the next input line,

galaxy called \error \ "0"o\c c,

something new will happen: T<sub>E</sub>X will type back an error message, saying

! Undefined control sequence.

(\*) galaxy called \error

\ "0"o\c c,

↑

What does this mean? It means, as you might guess, that an undefined control sequence was found in the input. T<sub>E</sub>X shows how far it has read your input by displaying it in two lines; the first line shows what has been read before the error was detected (namely "galaxy□called□\error□") and the next line shows what T<sub>E</sub>X hasn't looked at yet but will see next. So it is plain that "`\error`" is the culprit; it is a control sequence that hasn't been defined. After an error message, all is not lost, you have several options:

(1) Type `(line-feed)`. This will cause future error messages to be printed on your terminal as usual, but `TEX` will always proceed immediately without waiting for your response. It is a fast, but somewhat dangerous, way to proceed.

(2) Type `"x"` or `"X"`. This will cause `TEX` to stop right then and there, but you will be able to print any pages that have been completed.

(3) Type `"e"` or `"E"`. This will terminate `TEX` and activate the system editor, allowing you to edit the input file that `TEX` is currently reading from, if any.

(4) Type `"i"` or `"I"`. This will cause `TEX` to prompt you (with `"*`") for text to be *inserted* at the current place in the input; `TEX` will go on to read this new text before looking at what it ordinarily would have read next. You can often use this option to fix up the error. For example, if you have misspelled a control sequence, you can simply insert the correct spelling. (The `(carriage-return)` that you type after an insertion does not count as a space in the inserted text.)

(5) Type `(carriage-return)`. This is what you should do now. It causes `TEX` to resume its processing.

(6) Type a number (1 to 9). `TEX` will delete this many tokens from the input that it ordinarily would have read next, and then it will come back asking you to choose one of these options again. (A "token" is a single character or a control sequence. In certain rare circumstances `TEX` will not carry out the deletions, but you probably will never run into such cases.)

(7) Type `"?"` or anything else. Then `TEX` will refresh your memory about options (1) to (6), and will wait again for you to exercise one of these options.

If you respond by `(carriage-return)` or `(line-feed)` or `"i"` or `"I"`, `TEX` tries to recover from the error as best it can before carrying on. For example, `TEX` simply ignores an undefined control sequence like `\error`. If the error message is

```
! Missing } inserted.
```

`TEX` has inserted a `}` which it has reason to believe was missing. Chapter 27 discusses error messages and appropriate recovery procedures in further detail.

OK, you were supposed to type this line containing an `\error` so that you could experience the way `TEX` sometimes complains at you. Similar incidents will probably happen again, since `TEX` is constantly on the lookout for mistakes. The program tries to be a helpful and constructive critic, to catch errors before

they lead to catastrophes. But sometimes, like all programs, it really doesn't understand what's going on, so you have to humor it a bit.

On the remainder of the `\error` line you will note the strange concoction

```
\`O\`o\c c
```

and you already know that `\`` stands for an umlaut accent. The `\c` stands for a "cedilla" accent, so you will get

```
Ööç
```

as the name of that distant galaxy.

The next two lines are very simple, except that we haven't encountered `\par` before. This is one of the ways to end a paragraph. (Another way is to have a completely blank line. A third way is to come to the end of a file-page in an input file.)

The following lines of the example are also quite straightforward; they provide a review of the conventions we discussed long ago for dashes and quotation marks.

But when you type `\end` in the position shown, you will get another error message. The `\end` instruction is the normal way to stop T<sub>E</sub>X, but it has to occur at a proper time: not in mid-paragraph. The error message you get this time is

```
! You can't do that in horizontal mode.
```

As we will see later, T<sub>E</sub>X gets into various "modes," and it is in "horizontal mode" when it is making a paragraph. If you try to do something that is incompatible with the current mode, you will get this sort of error message. The proper response here is, once again, to hit (carriage-return); T<sub>E</sub>X will resume and forget that you said `\end` when you shouldn't.

The final line of the example says `\par` (to end the paragraph and get you out of horizontal mode), then it says

```
\vfill
```

(which means vertical fill—it will insert as much space as necessary to fill up the current page), then it says

```
\end
```

and now  $\text{\TeX}$  will end its processing gracefully. An "xspool" command will appear on your terminal; just hit (carriage-return) and the XGP will print your output. (At least, this is what will happen if you are at Stanford using the WAITS system.)

The output corresponding to the above example will not be shown in this manual; you'll have to do the experiment personally in order to see what happens.

At this point you might also like to look at the file called `ERRORS.TMP` on your area, since it records the error messages that  $\text{\TeX}$  typed back at you. Say "type errors.tmp" to the operating system.

►**Exercise 6.1:** If you had typed the second line of the story as

```
galaxy called \"0\"o\cc,
```

$\text{\TeX}$  would have issued an error message saying that the control sequence `\cc` is undefined. What is the best way to recover from this error?

That was Experiment Number 1, and you're ready for Experiment Number 2—after which you will be nearly ready to go on to the preparation of large manuscripts.

For Experiment 2, prepare a file called `STORY.TEX` that contains all the lines of the above example from "`\vskip 1 in`" to "`\par\vfill\end`" inclusive; but change the last line to

```
\par\vfill\eject
```

instead. (The `\eject` instruction is something like `\end`; it ends a page, but not the whole job.) Note that the line that specifies `\hsize` is to be omitted from your `STORY` file; the reason is that we are going to try typesetting the same story with a variety of column widths.

Start  $\text{\TeX}$  again (`r tex`), and `\input basic` again. But now type

```
\hsize 4 in
\input story
```

and see what happens. Guess what:  $\text{\TeX}$  is now going to set 4-inch columns, and it is going to read your `STORY.TEX` file.

Again it is going to hiccup on the undefined control sequence `\error`. This time try typing "e", so you can see how to get right to the system file editor from TeX in case your file is messed up. Delete the offending `\error` from the file, then start TeX off from scratch again.

Now try typing several instructions on the same line:

```
\input basic\hsize 4in\input story
```

If you don't put a blank space after the `c` of `basic` here, you'll get an error message (a file name should be followed by a blank space), but in this case it's safe to hit (carriage-return) and `continuc`. (TeX is just warning you that something may have been amiss; the rule is that a space should be there, but it will be inserted if you proceed. From now on, always leave a space after file names, to avoid any hassle.)

Soon TeX will be reading your `story` file again—and it will hang up on the `\end` error. Instead of removing this error, just type (line-feed) since you know it is harmless to bypass this error.

When TeX asks for more input, type the following lines, one at a time:

```
\hsize 3in \input story
\hsize 1.5in \input story
\jpar 1000 \input story
\ragged 1000 \input story
\hsize 1 in \input story
\end
```

The results will be somewhat interesting, so try it!

If you have followed instructions, your output will consist of six pages; the first page has MY STORY set 4 inches wide, the next has it set 3 inches wide, then come three pages where it is set  $1\frac{1}{2}$  inches wide, and a final page where TeX tries to make 1-inch columns. Since 1-inch columns of 10-point type allow only about 15 characters per line, the last four pages put quite a strain on TeX's ability to break paragraphs up into attractive lines.

When TeX fails to find a good way to handle a paragraph, there usually is no good way (except that TeX doesn't know how to hyphenate all words). In such cases the symptom is that TeX reports an "overfull box," and lines that are too



long will appear in the output. You probably noticed such a complaint about overfull boxes when  $\TeX$  was first trying to set the story with 1.5 inch columns. (If you didn't notice it on your terminal, look at `errors.tmp` to refresh your memory.) Several lines on page 3 of your output will be more than 1.5 inches long—they are “overfull” and stick out like sore thumbs.

There are two remedies for overfull boxes: You can either rewrite the text of the manuscript to avoid the problem (in fact, careful authors often do just that), or you can tell  $\TeX$  to consider larger spaces acceptable. The instruction `\jpar 1000` essentially makes  $\TeX$  look for more ways to break the paragraph, including those with larger spaces; so the fourth page of the output shows a solution of the problem without any overfull boxes.

② The expandability of spaces is defined by the font, not by  $\TeX$ . Standard  $\TeX$  fonts like `cmr10` have fairly tight restrictions on spacing, in accordance with the recommendations of contemporary typographers. These strict standards are appropriate for books, but not for newspapers, when more tolerance is needed. If you are setting a lot of material with narrow margins, it would be better to use a font with more variability in its spacing than to use a high setting of `\jpar`, since  $\TeX$  has to work harder when `\jpar` is large (it considers more possibilities). Chapter 14 explains more about `\jpar`.

② The instruction `\ragged 1000` causes paragraphs to be set with a “ragged right margin”—i.e., the lines are broken as usual, but spaces between words don't stretch or shrink very much. Chapter 14 tells more about `\raggedness`.

② When `\hspace` was one inch in the above experiment,  $\TeX$  again came up with an overfull box, even when `\jpar` was quite large. The reason is that  $\TeX$  doesn't know how to hyphenate “Drofnats”, the second word of the second paragraph. To remedy this, replace “Drofnats” by “Drof\–nats” in both places where it occurs in your `story` file, and try setting the story with

```
\hspace 1 in \jpar 1000 \ragged 0 .
```

You'll see that the output is now quite reasonable, considering the extremely narrow column width. The control sequence `\–` means a *discretionary hyphen*, namely a legal place to hyphenate the word if  $\TeX$  needs to.

At this point you might want to play around with  $\TeX$  a bit before you read further. Try different stories, different measurements, and so on. One experiment particularly recommended is to type

```
\ctrlline{MY \ERROR STORY}
```

after `basic` has been `\input`. This produces a somewhat more elaborate error message with which you should become acquainted, namely:

```
! Undefined control sequence.
<argument> MY \Error
          STORY
... plus1000cm minus1000cm #1
                                \hskip Opt plus1000cm minu...
(*) \ctrline{MY \ERROR STORY}
```

The reason for all this is that `\ctrline` is not a built-in T<sub>E</sub>X instruction, it is a control sequence defined in the `basic` format. Thus T<sub>E</sub>X did not detect any mistake when it read "`{MY \ERROR STORY}`", it simply absorbed this group and passed the text "`MY \ERROR STORY`" as an argument to the `\ctrline` definition. According to Appendix B, `\ctrline` gets expanded into the text

```
\hjust to size{\hskipOpt plus1000cm minus1000cm
                #1\hskipOpt plus1000cm minus1000cm}
```

where the argument gets inserted in place of the "`#1`". (You don't have to understand exactly what this means, just believe that it is a way to center something on a line.) A fragment of this expansion is shown in the error message, preceded and followed by "`...`" to indicate that there was more to the expansion T<sub>E</sub>X was reading. The error message shows that T<sub>E</sub>X had read the expansion up to the point "`#1`", because `\hskip` etc. appears on the next line. Furthermore the error message shows that T<sub>E</sub>X was reading the argument, and the last thing it read was the control sequence "`\Error`". (You actually typed "`\ERROR`", but upper case and lower case are not distinguished by T<sub>E</sub>X after the first letter of a control sequence.)

The point is that when you make an error within a routine controlled by a defined control sequence like `\ctrline`, the error message will show everything T<sub>E</sub>X knows about what it was reading; the display occurs in groups of two lines per level of reading, where the first line shows what T<sub>E</sub>X has read at this level and the second line shows what is yet to be read. Somewhere in there you should be able to spot the problem, the thing T<sub>E</sub>X wasn't expecting.

Ⓛ Careful study of the 1.5-inch example shows that T<sub>E</sub>X does not automatically break lines just before a dash, although it does do so just *after* one. Some printers will start new lines with dashes; if you really want to do this you can type "`\penalty 0`" just before each dash. For example, "`Drofmate\penalty 0---`".

### <7> How $\TeX$ reads what you type

While studying the example in the previous chapter, we observed that an input manuscript is expressed in terms of "lines" ending with (carriage-return)s, but these lines of input are essentially independent of the lines of output that will appear on the finished pages. Thus you can stop typing a line of input at any convenient place. A few other related rules have also been mentioned:

- A (carriage-return) is like a space.
- Two spaces in a row count as one space.
- A blank line denotes end of paragraph.

Strictly speaking, these rules are contradictory: A blank line is obtained by typing (carriage-return) twice in a row, and this is different from typing two spaces in a row. So now let's see what the *real* rules are. The purpose of this chapter is to study the very first stage in the transition from input to output.

In the first place, it's wise to have a precise idea of what your keyboard sends to the machine. There are 128 characters that  $\TeX$  might encounter at each step in a file or in a line of text typed directly on your terminal. These 128 characters are classified into 13 categories numbered 0 to 12:

Category code	Meaning
0	Escape character ( $\backslash$ in this manual)
1	Beginning of group ( $\{$ in this manual)
2	End of group ( $\}$ in this manual)
3	Begin or end math ( $\$$ in this manual)
4	Alignment tab ( $\otimes$ in this manual)
5	End of line ((carriage-return) and $\%$ in this manual)
6	Parameter ( $\#$ in this manual)
7	Superscript ( $\uparrow$ in this manual)
8	Subscript ( $\downarrow$ in this manual)
9	Ignored character
10	Space
11	Letter (A, ..., Z and a, ..., z)
12	Other character

It's not necessary for you to learn these code numbers; the point is only that  $\TeX$  responds to 13 different types of characters. At first this manual led you to

believe that there were just two types—the escape character and the others—and more recently you were told about two more types, the grouping symbols like { and }. Now you know that there are really 13. This is the whole truth of the matter; no more types remain to be revealed.

Actually no characters are defined to be of types 0 to 8 when TeX begins, except that `<carriage-return>` and `<form-feed>` are type 5. But if you are using a predefined format (like almost everybody does) you will be told which characters have special significance. For example, if you are using the basic package of Appendix B you need to know that the nine characters

\ { } \$ @ % # † ‡

cannot be used as ordinary characters in your text; they have special meaning. (If you really need any of these symbols as part of what you're typing, e.g., if you need a \$ to represent dollars, there is a way out—this will be explained later. A list of control sequences for special symbols appears in Appendix F.)

When TeX is reading a line of text from a file, or a line of text that you entered directly on your terminal, it is in one of three "states":

State <i>N</i>	Beginning a new line
State <i>M</i>	Middle of a line
State <i>S</i>	Skipping blanks

At the beginning it's in state *N*, but most of the time it's in state *M*, and after a control sequence or a space it's in state *S*. Incidentally, "states" are different from the "modes" mentioned in Chapter 6; the current *state* refers to TeX's eyes and mouth as they take in characters of new text, but the current *mode* refers to the condition of TeX's gastro-intestinal tract. Modes are discussed further in Chapter 13.

You hardly ever need to worry about what state TeX is in, but you may want to understand the rules just in case TeX does something unexpected to your input file. In general, it is nice to understand who you are talking to.

Furthermore, if you faithfully carried out the experiment in the previous chapter you will probably have noticed that there was an unwanted space after the dash in "called---"; the `<carriage-return>` after this dash got changed into a space that doesn't belong there. This error was purposely put into the example

because the author of this manual feels that we learn best by making mistakes. But now let's look closely into  $\text{\TeX}$ 's reading rules so that such mistakes will be unlearned in the future.

Fortunately the rules are not complicated or surprising; you could probably write them down yourself:

If in state  $N$  (new line) and  $\text{\TeX}$  sees

- a) an escape character (type 0),  $\text{\TeX}$  scans the entire control sequence, then digests it (i.e., sends the control sequence to the guts of  $\text{\TeX}$  where it will be processed appropriately) and goes to state  $S$ .
- b) an end-of-line character (type 5),  $\text{\TeX}$  throws away any other information that might remain on the current line, then digests a " $\backslash\text{par}$ " instruction (paragraph end) and remains in state  $N$ .
- c) an ignored character or a space (types 9,10),  $\text{\TeX}$  passes it by, remaining in state  $N$ .
- d) anything else (types 1,2,3,4,6,7,8,11,12),  $\text{\TeX}$  digests it and goes to state  $M$ .

In summary, when  $\text{\TeX}$  is beginning a line, it skips blanks, and if it gets to the end of the line without seeing anything it considers that a paragraph has ended.

If in state  $M$  (middle of line) and  $\text{\TeX}$  sees

- a) an escape character (type 0),  $\text{\TeX}$  scans the entire control sequence, then digests it and goes to state  $S$ .
- b) an end-of-line character (type 5),  $\text{\TeX}$  throws away any other information that might remain on the current line, then digests a blank space and goes to state  $N$ .
- c) an ignored character (type 9),  $\text{\TeX}$  passes it by, remaining in state  $M$ .
- d) a space (type 10),  $\text{\TeX}$  digests a blank space and goes to state  $S$ .
- e) anything else (types 1,2,3,4,6,7,8,11,12),  $\text{\TeX}$  digests it and remains in state  $M$ .

In summary, when  $\text{\TeX}$  is in the middle of a line, it digests what it sees, but converts one or more blank spaces into a single blank space, and also treats the end of line as a blank space.

If in state *S* (skipping blanks) and TeX sees

- a) an escape character (type 0), TeX scans the entire control sequence, then digests it, remaining in state *S*.
- b) an end-of-line character (type 5), TeX throws away any other information that might remain on the current line, then switches to state *N*.
- c) an ignored character or a space (types 9,10), TeX passes it by, remaining in state *S*.
- d) anything else (types 1,2,3,4,6,7,8,11,12), TeX digests it and goes to state *M*.

In summary, when TeX is skipping blanks, it ignores blanks and doesn't treat the end of a line as a blank space.

So those are the rules. Only three major consequences deserve special emphasis here:

First, a `<carriage-return>` always counts as a space, even when it follows a hyphen. If you want to end a line with a `<carriage-return>` but no space, you can do this by typing the control sequence `"\!"` just before the `<carriage-return>`. For example, the 7th-last line of MY STORY in Chapter 6 should really have been typed as follows:

```
he preferred to be called---\!
```

A second consequence of the rules, if you are using the basic format of Appendix B, is that the % sign is treated as an end-of-line mark equivalent to a `<carriage-return>`. This is useful for putting comments into the manuscript. For example, you might include a copyright notice for legal protection; or you might say

```
% Figure 5 belongs here;
```

or you might say

```
% This } is supposed to match the { of "\ctrline{".
```

Anything that you might want to remember but not to print can be included after a %, because TeX will never look at the rest of the line.

A third consequence of the rules is that you should indicate the end of a paragraph either explicitly, by using the control sequence `\par`; or implicitly, by having an entirely blank line. (The end of a file page also counts as a blank line, because of the way files of text are conventionally represented in the computer.) In the latter case, `TeX` has always read a space before it came to the end of the paragraph, because it digested a space at the end of the line before the blank line. In the former case, you may or may not have typed a space before you typed `"\par"`. Fortunately, there's nothing to worry about; the result is the same in either case, because `TeX`'s paragraph processor discards the final item of a paragraph when it is a space.

If you have several blank lines in a row, `TeX` digests a `"\par"` instruction for each one, according to the rules. But this doesn't show up in the output, because empty paragraphs are discarded.

►**Exercise 7.1:** If a line isn't entirely blank, but the first nonblank character on the line is `%`, does this signify end-of-paragraph?

Ⓜ When `TeX` first starts up, the 128 possible characters are initially interpreted as follows. Characters "A" to "Z" (ascii codes '101 to '132) and "a" to "z" (ascii codes '141 to '172) are type 11 (letters). The characters `<null>`, `<line-feed>`, `<vertical-tab>`, `<alt-mode>`, and `<delete>` (ascii codes 0, '12, '13, '175, and '177 at Stanford) are type 9 (ignored). The characters `<tab>` and `< >` (ascii codes '11 and '40) are type 10 (spaces). The characters `<form-feed>` and `<carriage-return>` (ascii codes '14 and '15) are type 5 (end of line). All other characters are type 12 (other). The first non-space input by `TeX` is defined to be the escape character used in error messages, and it is set to type 0 (escape). You can use `\chcode` to change the type code of any character, and it is possible to have several characters each defined to be of type 0 or any other type. The instruction

```
\chcode<number1>+<number2>
```


(where `<number1>` is between 0 and 127 and `<number2>` is between 0 and 12) causes the character whose 7-bit code is `<number1>` to be regarded as type `<number2>` for the duration of the current group, unless its type is changed again by another `\chcode`. For example, if for some reason you want `TeX` to treat the letter "a" as a non-letter, you could say

```
\chcode'141+12 .
```

But this would probably not be useful because, e.g., `"\par"` would no longer be a control sequence; it would be read as `"\p"` followed by `"a"` followed by `"r"`.

We will see later that spaces are sometimes ignored after other things besides control sequences, since there are various  $\TeX$  constructions that look better if spaces or end-of-line follow them. For convenient reference, here is a list of all cases in which  $\TeX$  will ignore a space, even though most of these constructions haven't been explained yet in the manual:

- After a space or end-of-line character.
- After a control sequence.
- After the `}` that ends a `\def` or `\if` or `\ifeven` or `\else` or `\noalign` or `\output` or `\mark`.
- Between `$` signs, when  $\TeX$  is in math mode.
- After the `$$` that ends a display.
- After a file name or an already-defined font code or a unit of measure or the words "to" or "size" in justification specifications.
- Before or after a `<number>` or the sign preceding a `<number>`.
- After a paragraph, or in general whenever  $\TeX$  is in vertical mode or restricted vertical mode.

  $\TeX$  goes into reading state *S* only as shown in the detailed reading rules above. When it ignores spaces at other times, e.g. after a unit of measure, the spaces it ignores are actually "digested" spaces; the processing routine calls on  $\TeX$ 's input mechanism to continue reading until a non-space is digested. This is a fine point, because it hardly ever makes a difference; but here is a case where it matters: Suppose you make the definition "`\def\space{\ }`". Then if you type "`\space\space`",  $\TeX$  will digest two spaces; these spaces would not be ignored after a space or end of line or control sequence, because of  $\TeX$ 's reading rules, but they would be ignored in the other cases listed above, because of  $\TeX$ 's digestive processes. On the other hand `\`  (control space) is treated differently: it always means an explicit space and it is never ignored in any of the above cases except the last (in vertical mode). Sometimes  $\TeX$  will ignore only one digested space, but at other times it will ignore as many as are fed to it; if you really need to know which cases fall into each category, you can find out by experiment.

## <8> The characters you type

A lot of different keyboards are used with  $\TeX$ , but few keyboards can produce 128 different symbols. Furthermore, as we have seen, some of the characters that



you can type on your keyboard are reserved for special purposes like escaping and grouping. Yet when we studied fonts it was pointed out that there are 128 characters per font. So how can you refer to the characters that aren't on your keyboard, or that have been pre-empted for formatting?

One answer is to use control sequences. For example, the `basic` format of Appendix B, which defines `%` to be an end-of-line symbol so that you can use it for comments, also defines the control sequence `\%` to mean a per-cent sign.

To get access to any character whatsoever, you can type

```
\char⟨number⟩
```

where `⟨number⟩` is any number from 0 to 127 (optionally followed by a space), and you will get the corresponding character from the current font. For example, the letter "b" is character number 98, so you could typeset the word `bubble` by typing


```
\char98u\char98\char98le
```

if the `b`-key on your typewriter is out of order. (Of course you need the `\`, `c`, `h`, `a`, and `r` keys to type `"\char"`, so let's hope they are always working.)

Character numbers are usually given in *octal notation* in reference books (i.e., using the radix-8 number system). A `⟨number⟩` in `TeX`'s language can be preceded by a `^`, in which case it is understood as octal. For example, the octal code for "b" is `142*`, so

```
\char^142
```

is equivalent to `\char98`. In octal notation, character numbers run from `^0` to `^177`.

 Formally speaking, a `⟨number⟩` in a `TeX` manuscript is any number of spaces followed by an optional `"^"` followed by any number of digits followed by an optional space. Or it can be any number of spaces followed by `"\count⟨digit⟩"` followed by an optional space; in the latter case the specified counter is used (cf. Chapter 23).

You can't use `\char` in the middle of a control sequence, though. If you type

```
\char^142
```

---

\*The author of this manual likes to use italic digits to denote octal numbers, instead of using the `^` symbol, when octal numbers appear in printed books.

$\TeX$  reads this as the control sequence `\c` followed by `h`, `a`, etc., not as the control sequence `\b`.

Actually you will hardly ever have to use `\char` yourself, since the characters you want will probably be available as predefined control sequences; `\char` is just a last resort in case you really need it (and it is also indispensable for the designers of book formats).

Since  $\TeX$  is intended to be useful on many different kinds of keyboards, it does not assume that you can type very many of the exotic characters. For example, if your keyboard has an  $\alpha$  on it (Greek lower case alpha)—this is character code 2 at Stanford—you will be able to type “ $\alpha$ ” in a math formula and get an alpha. But if you don't have  $\alpha$  on your keyboard,  $\TeX$  understands the control sequence `\alpha` just as well.

Character code 2 in  $\TeX$ 's font `cmr10` is not really an alpha; it is actually  $\Theta$ , an upper case Greek theta!  $\TeX$  doesn't want you to type “ $\alpha$ ” except in math formulas. When you are typing straight text with  $\TeX$ 's special fonts like `cmr10`, you should confine yourself to the symbols usually found on a typewriter and a few more that are listed in the next chapter. In fact, every font you use might have a different way of assigning its symbols to the numbers 0 to 127. Whoever designed the font should tell you what this encoding is. It's not even guaranteed that an “a” will yield an “a”. Your keyboard converts what you type into codes between 0 and 127, and these codes will select the corresponding characters of the current font, but a font designer can put whatever symbol he or she wants into each position.

Furthermore, *different fonts might also have different ligatures*. It isn't true that `--` will give you a dash in all fonts with  $\TeX$ , nor that ```` will become “”, nor that `ffl` will become `ffl`. Each font designer decides what ligature combinations will appear in his or her font, and this person should tell you what they are. The seven ligatures

`..` `..` `--` `---` `ff` `fi` `fl` `ffi` `ffl`

described in Chapter 2 are available in all the “standard”  $\TeX$  roman and slanted fonts, but you should not assume that they are present in all fonts.

Similarly, accents like `\`` and `\` can't be used with all fonts; the accent characters have to be in certain positions within the font, and not all fonts have them.

⚠ If you want to use an accent on a nonstandard font (e.g., if you need a new accent for some newly discovered African dialect), suppose you have a font that includes this accent as character number '20. Then you can type "`\accent'20a`" to get this accent over an "a", etc. In general, type

$$\backslash\text{accent}\langle\text{number}\rangle\langle\text{char}\rangle$$

to get an accent over a character in the same font, or

$$\backslash\text{accent}\langle\text{number}\rangle\backslash:\langle\text{font}\rangle\langle\text{char}\rangle$$

to get an accent over a character in a different font. You're not allowed to say things like "`{\:\:b\backslashaccent'20}a`", however; the character to be accented must immediately follow the accent except for font changes.

### <9> T<sub>E</sub>X's standard roman fonts

When you are using a standard roman font (like `cmr10`, `cmb10`, `cms10`, or `cmss10`, which stand respectively for Computer Modern Roman, Bold, Slanted, or Sans-Serif, 10 points high), you need to know the information in this chapter.

These fonts are intended to contain nearly every symbol you will need for non-math text, including accents and special characters for use with foreign languages. When you are using such fonts you should confine yourself to typing the following symbols only:

the letters A to Z and a to z

the digits 0 to 9

the standard punctuation marks , : ; ! ? ( ) [ ] & ` ' - \* / .

You can also type `+ = < >` and you will get the corresponding symbols, but this is not recommended because these symbols should be used only in mathematics mode (explained later). The result will look better in mathematics mode, because T<sub>E</sub>X will insert proper spacing. When you use the `"-`" and `"/`" it should not be for mathematics; do hyphens and slashes outside of math mode, but don't do subtractions and divisions.

Conspicuously absent from this list are the following symbols found on many keyboards:

`\ { } # $ % ^ _ " @`

Resist the temptation to type them. Also resist the temptation to type mathematical symbols like

`| ← α β ε λ π √ ∃ ∞`

and so on, if your keyboard has them. Like + and =, they should be reserved for mathematics mode; but unlike + and =, they don't give the results you might expect, except in mathematics mode.

By using control sequences you can obtain the following special symbols needed in foreign languages:

Type	to get
<code>\ss</code>	ß (German letter ss)
<code>\ae</code>	æ (Latin and Scandinavian ligature ac)
<code>\AE</code>	Æ (Latin and Scandinavian ligature AE)
<code>\oe</code>	œ (French ligature oe)
<code>\OE</code>	Œ (French ligature OE)
<code>\o</code>	ø (Scandinavian slashed o)
<code>\O</code>	Ø (Scandinavian slashed O)

For example, if you want to specify "Æsop's Œuvres en français" you could type

```
\AE sop's \OE uvres en fran\c cais .
```

(Note the spaces after these control sequences. Another way to separate them from the surrounding text would be

```
{\AE}sop's {\OE}uvres en fran{\c c}ais ;
```

this looks a little nicer, perhaps, in the computer file, but it's harder to type.)

The following accents are available in standard roman fonts, shown here with the letter "o":

Type	to get
<code>\`o</code>	ò (accent grave)
<code>\´o</code>	ó (accent aigu, acute accent)
<code>\A o</code>	ô (accent circonflexe, circumflex or "hat" accent)
<code>\v o</code>	ǎ (Slavic accent, inverted circumflex)
<code>\u o</code>	ø (breve, short vowel)
<code>\=o</code>	ō (macron or bar, long vowel)
<code>\"o</code>	ö (umlaut or double dot)
<code>\H o</code>	ő (long Hungarian umlaut)
<code>\b o</code>	õ (vector accent—used in mathematics)
<code>\s o</code>	õ (tilde or squiggle)
<code>\t oo</code>	oo (ties two letters together)
<code>\a a</code>	å (Scandinavian a with circle)
<code>\l l</code>	ł (Polish crossed l)
<code>\c c</code>	ç (cedilla accent)

The last three of these examples are shown with other letters instead of "o" because they are somewhat special; the Scandinavian accent is shown over an "a" since "ö" isn't a Scandinavian letter. Similarly, the `\l` accent is specifically designed for the letter "l". Cedillas are usually associated with the letter "c" (although it is true that "ç" appears in Navajo).

Spaces are obligatory where shown in these examples. But the space can be omitted after the accent codes `\``, `\´`, `\=`, and `\"`, since they don't involve letters.

Within a font, accents are designed to appear at the right height for letters like "o"; but `TeX` will raise an accent if it is applied to a tall letter. For example, the result of `"\`O"` is "Ö". This simple rule almost always works all right, but sometimes it fails; for example, an upper case A with the circle accent traditionally has the circle touching the A ( $\text{\AA}$ ), at least in Scandinavian books, while `"\a A"` yields "Å". (Both of these forms are used by modern American printers to denote angstrom units, but Å is preferable.) The `\l` doesn't work with a capital L either; `"\l L"` yields "L". An even more conspicuous failure of `TeX`'s rule occurs if you try to put a cedilla on an upper case "C" by typing `"\c C"`; `TeX` will raise the cedilla to give "C"! (See below for how to handle these anomalous cases.)

When the letters "i" and "j" are accented, it is traditional to omit the dots they contain. Therefore standard roman fonts contain the dotless letters

i and j

which you can obtain by typing "\i" and "\j", respectively. For example, to obtain "mīnūs" you would type "m\=\i n\u us".

►**Exercise 9.1:** Explain what to type in order to get the sentence

*Commentarii Academæ Petropolitanae* is now *Doklady Akademiâ Nauk SSSR*.

►**Exercise 9.2:** How would you specify the names Øystein Ore, ÎUri ÎAnov, Ja'far al-Khowârizmî, and Władysław Süßman?

Ⓓ The character to be accented must immediately follow the accent, except for the fact that you are allowed to change fonts in between; see the remarks at the close of the previous chapter. T<sub>E</sub>X adjusts for the slantedness of characters when placing accents, including the possibility that the accent comes from a font with a different slant than the character being accented. For example, if you type

```
\`e \`E \sl\`e \`E \rm\`sl e \rm\`sl E \`rm e \sl\`rm E
```

using `basic` format, the result will be

é É é É é É é É.

Ⓓ The fonts are designed so that the anomalous cases of "bad accents" mentioned above can be handled as follows, using the `\spose` (superpose) control sequence of `basic` format: To get

À Ç Ì

type respectively

```
\spose{\raise 1.667pt\hjust{\char'27}}A
\spose{\char'30}C
\spose{\raise 2.5pt\hjust{\char'31}}L
```

(This is for 10-point sizes; the amounts to raise the accents must be adjusted proportionately when working with other sizes. For example, "`\raise 1.667pt`" would become "`\raise 1.5pt`" in 9-point type.)

A complete list of the 128 symbols in T<sub>E</sub>X's standard roman fonts appears in Appendix F. But everything a typist needs to know about them has already been explained; it's not necessary for you to know the numeric character codes.

**<10> Dimensions**

The example program used in the trial runs of Chapter 6 involved mysterious  $\TeX$  instructions like “\vskip 2.54cm”. Now it is time to reveal part of this mystery, by explaining what units of measure  $\TeX$  understands.

“Points” and “picas” are printers' traditional basic units of measure, so  $\TeX$  understands points and picas.  $\TeX$  also understands inches and certain metric units, but it converts everything internally to points. Each unit of measure is given a two-letter abbreviation; here is a complete list of the units  $\TeX$  knows about:

```
pt  point
pc  pica (one pica equals 12 points)
in  inch (one point equals 0.01383700 inches)
cm  centimeter (one inch equals 2.5400 centimeters)
mm  millimeter (one centimeter equals 10 millimeters)
dd  Didot point (one centimeter equals 28.600 Didot points)
```

When you want to express some physical dimension to  $\TeX$ , type it as

(optional sign)(number)(unit of measure)

or

(optional sign)(number) . (number)(unit of measure)

(and in the second case your (number)s had better not be in octal notation or  $\TeX$  will get confused). An (optional sign) is either a “+” or a “-” or nothing at all.

For example, here are some typical lengths:

```
3 in
29 pc
-0.013837in
+ 42.1 dd
0 mm
```

A plus sign is redundant, but some people like occasional redundancy.


Spaces are optional before and after numbers and after the units of measure, but you should not put spaces *within* a number or between the two letters in the unit of measure.

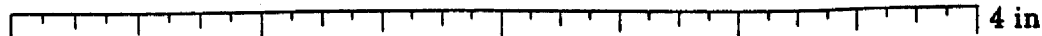
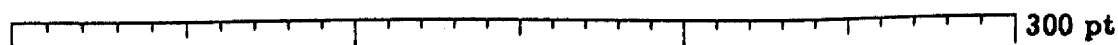
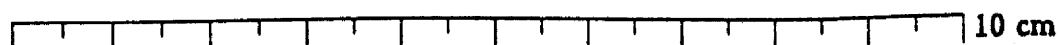
In a manual like this it is convenient to use "angle brackets" in abbreviations for various constructions like  $\langle$ number $\rangle$  and  $\langle$ optional sign $\rangle$ . Henceforth in this manual we will use the term  $\langle$ dimen $\rangle$  to stand for any dimension expressed in the above form. For example,

$$\backslash\text{hsize}\langle\text{dimen}\rangle$$

will be the general way to define the page width  $\text{T}_{\text{E}}\text{X}$  is supposed to use.

When a dimension is zero, you have to specify a unit of measure even though it is redundant. Don't just say "0", say "0pt" or "0in" or something.

 Chapter 6 mentions that units of measure may be inflated artificially on some output devices. The following "rulers" have been typeset by  $\text{T}_{\text{E}}\text{X}$  so that you can calibrate the output device used to produce the copy of the manual you are reading:

►**Exercise 10.1:** (To be worked after you know about boxes and glue and have read Chapter 21.) Explain how to typeset a 10 cm ruler like this using  $\text{T}_{\text{E}}\text{X}$ .

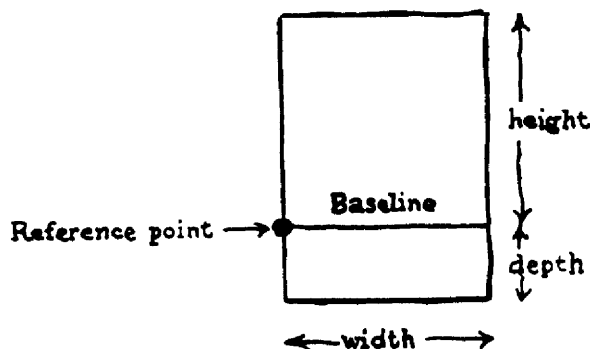
## <11> Boxes

$\text{T}_{\text{E}}\text{X}$  makes complicated pages by starting with simple individual characters and putting them together in larger units, and putting these together in still larger units, and so on. Conceptually, it's a big paste-up job. The  $\text{T}_{\text{E}}\text{X}$  terms used to describe such page construction are *boxes* and *glue*.

Boxes in  $\text{T}_{\text{E}}\text{X}$  are two-dimensional things with a rectangular shape, having three associated measurements called *height*, *width*, and *depth*. Here is a picture



of a typical box, showing its so-called reference point and baseline:



From  $\text{T}_{\text{E}}\text{X}$ 's viewpoint, a single character from a font is a box, one of the simplest kinds of boxes. The font designer has decided what the height, width, and depth of the character are, and what the symbol will look like when it is in the box;  $\text{T}_{\text{E}}\text{X}$  just uses these dimensions to paste boxes together, and ultimately to determine the locations of the reference points for all characters on a page. In the `cmr10` font, for example, the letter "h" has a height of 6.9444 points, a width of 5.5556 points, and a depth of zero; the letter "g" has a height of 4.4444 points, a width of 5 points, and a depth of 1.9444 points. Only certain special characters like parentheses have height plus depth actually equal to 10 points, although `cmr10` is said to be a "10 point" font. The typist doesn't have to know these measurements, of course, but it is helpful for  $\text{T}_{\text{E}}\text{X}$ 's users to be aware of the sort of information  $\text{T}_{\text{E}}\text{X}$  deals with.

The character shape need not fit inside the boundaries of the box. For example, some characters that are used to build up larger symbols like square-root signs intentionally protrude a little bit, so that they overlap properly with the rest of the symbol. Slanted letters frequently extend a little to the right of the box, as if the box were skewed right at the top and left at the bottom, keeping its baseline fixed. For example, compare the letter "q" in `cmr10` and `cms10` fonts:



In both cases  $\text{T}_{\text{E}}\text{X}$  thinks the box is 5 points wide, so both letters get exactly the same treatment.  $\text{T}_{\text{E}}\text{X}$  doesn't know exactly where the ink will go—only the font

designer knows this. But the slanted letters will be spaced properly in spite of  $\TeX$ 's lack of knowledge, because the baselines will match up.

Actually the font designer also tells  $\TeX$  one other thing, the so-called *italic correction*: A number is specified for each character, telling roughly how far that character extends to the right of its box boundary. For example, the italic correction for "q" in `cmr10` is zero, but in `cms10` it is 0.2083 points. If you type the control sequence

\/

following a character,  $\TeX$  will effectively increase the width of that character by the italic correction. It's a good idea to use `\/` when shifting from slanted to unslanted fonts without intervening spaces, for example when a slanted word is immediately followed by an unslanted right parenthesis or semicolon. The author typed

the so-called `{\sl italic correction\/}`:

when specifying the first sentence of the paragraph you are now reading. Of course, there's no need to make the italic correction when a slanted letter is followed by an unslanted period or comma.


Another simple kind of box  $\TeX$  deals with might be called a "black box," a rectangle like "■" that is to be entirely filled with ink at printing time. You can specify any height, width, and depth you like for such boxes—but they had better not have too much area or the printer might get upset. (Printers generally prefer white space to black space.)


Usually these black boxes are made very skinny, so that they appear as horizontal lines or vertical lines. Printers traditionally call such lines "horizontal rules" and "vertical rules," so the terms  $\TeX$  uses to stand for black boxes are `\hrule` and `\vrule`. We will discuss the use of rule boxes in greater detail later.

Everything on a page that has been typeset by  $\TeX$  is made up of simple character boxes or rule boxes, pasted together in combination.  $\TeX$  pastes boxes together in two ways, either *horizontally* or *vertically*. When  $\TeX$  builds a horizontal list of boxes, it lines them up so that their reference points appear in the same horizontal row; therefore the baselines of adjacent characters will match up as they should. Similarly, when  $\TeX$  builds a vertical list of boxes, it lines them up so that their reference points appear in the same vertical column.


There is also a provision for lowering or raising the reference points of individual boxes in a horizontal list. This has been used, for example, to lower the

"E" in "T<sub>E</sub>X". Similarly, there is a way to move the reference points of boxes to the left or to the right in a vertical list. This is used, for example, when centering an accent over a letter, since an accented letter like  $\acute{E}$  is essentially a box made from a vertical list containing the two character boxes "" and "E".

 When a big box has been made from a horizontal list of smaller boxes, the baseline of the big box is the common baseline of the smaller boxes. (More precisely, it's the common baseline they would share if they hadn't been raised or lowered.) The height and depth of the big box are determined by the maximum distances that the smaller boxes reach above and below the baseline, respectively; any raising and lowering of the smaller boxes is taken into account during this calculation. The width of the big box is determined by whatever T<sub>E</sub>X operation was used to create that box, as explained in the next chapter.

 When a big box has been made from a vertical list of smaller boxes, its reference point is the reference point of the last (lowest) box in the list (but ignoring left or right shifts). The depth of the big box is therefore equal to the depth of this last smaller box. The width of the big box is determined by the maximum distance that the smaller boxes reach to the right of the reference point; any left or right shifting of the smaller boxes is taken into account during this calculation. (Note that if any of the smaller boxes have been shifted left, they will protrude past the left boundary of the big box.) The height of the big box is determined by whatever T<sub>E</sub>X operation was used to create that box, as explained in the next chapter.

A page of text like the one you're reading is itself a box, in T<sub>E</sub>X's view: It is a largish box made from a vertical list of smaller boxes representing the lines of text. Each line of text, in turn, is a box made from a horizontal list of boxes representing the individual characters. In more complicated situations, involving mathematical formulas and/or complex tables, you can have boxes within boxes within boxes ... to any level. But even these complicated situations arise from horizontal or vertical lists of boxes pasted together in a simple way, so all that you and T<sub>E</sub>X have to worry about is one list of boxes at a time. In fact, when you're typing straight text, you hardly have to think about boxes at all, since T<sub>E</sub>X will automatically take responsibility for assembling the character boxes into words and the words into lines and the lines into pages. You only need to be aware of the box concept when you want to do something out of the ordinary, like centering a heading or providing extra space, etc.

 The height, width, or depth of a box might be negative, in which case it is a "shadow box" that is somewhat hard to draw. You might be able to think of some

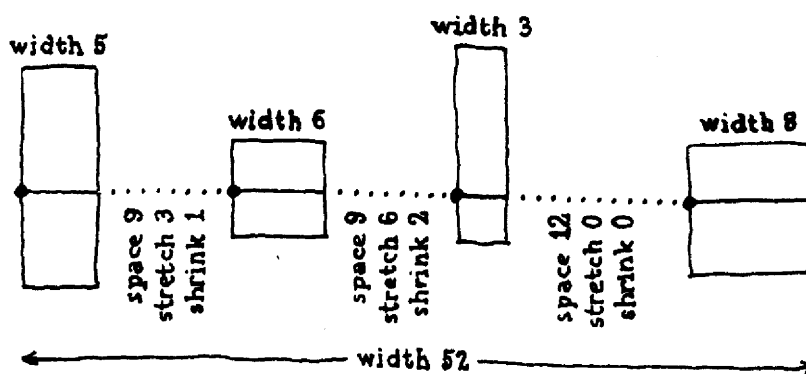
tricky things to do with such boxes;  $\TeX$  just lines things up and adds up dimensions as if everything were positive or zero. Thus, for example, if a font designer specified a character with negative width, it would act like a backspace. When forming a box from a horizontal list, however,  $\TeX$  sets the height and depth to zero if they turn out to be negative, so only the width can be negative. Similarly, only the height and depth of a box formed from a vertical list can be negative. Negative dimensions are not allowed in rule boxes.

### <12> Glue

But there's more to the story than just boxes: there's also some magic mortar called *glue* that  $\TeX$  uses to paste boxes together. For example, there is a little space between the lines of text in this manual; it has been calculated so that the baselines of consecutive lines within a paragraph are exactly 12 points apart. And there is space between words too; such space is not an "empty" box, it is part of the glue between boxes. This glue can stretch or shrink so that the right margin of each page comes out looking straight.

When  $\TeX$  makes a large box from a horizontal or vertical list of smaller boxes, there often is glue between the smaller boxes. Glue has three attributes, namely its natural *space*, its ability to *stretch*, and its ability to *shrink*.

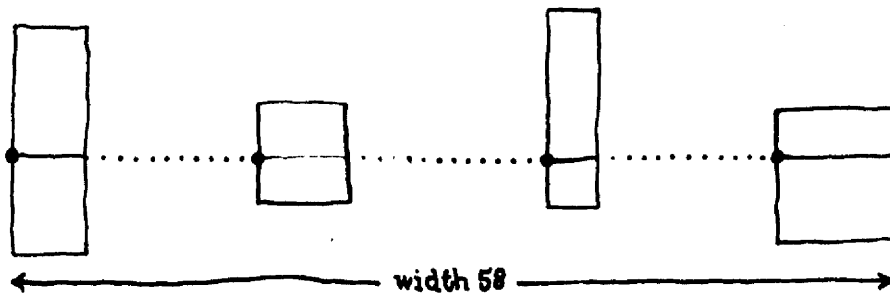
In order to understand how this works, consider the following example of four boxes in a horizontal list separated by three globs of glue:



The first glue element has 9 units of space, 3 of stretch, and 1 of shrink; the next one also has 9 units of space, but 6 units of stretch and 2 of shrink; the last one

has 12 units of space, but it is unable to stretch or to shrink, so it will remain 12 units of space no matter what.

The total width of boxes and glue in this example, considering only the space components of the glue, is  $5 + 9 + 6 + 9 + 3 + 12 + 8 = 52$  units. This is called the *natural width* of the horizontal list; it's the preferred way to paste the boxes together. Suppose, however, that  $\text{\TeX}$  is told to make the horizontal list into a box that is 58 units wide; then the glue has to stretch by 6 units. Well, there are  $3 + 6 + 0 = 9$  units of stretchability present, so  $\text{\TeX}$  multiplies each unit of stretchability by  $6/9$  in order to obtain the extra 6 units needed. Thus, the first glob of glue becomes  $9 + (6/9) \times 3 = 11$  units wide, the next becomes  $9 + (6/9) \times 6 = 13$  units wide, the last remains 12 units wide, and we obtain the desired box looking like this:



On the other hand, if  $\text{\TeX}$  is supposed to make a box 51 units wide from the given list, it is necessary for the glue to shrink by a total of one unit. There are three units of shrinkability present, so the first glob of glue would shrink by  $1/3$  and the second by  $2/3$ .

The process of determining glue thickness when a box is being made from a horizontal or vertical list is called *setting the glue*. Once glue has been set, it becomes rigid—it won't stretch or shrink any more, and the resulting box is essentially indecomposable.

Glue will never shrink more than its stated shrinkability. The first glob of glue above, for example, will never be allowed to become narrower than 8 units wide, and  $\text{\TeX}$  will never shrink the given horizontal list to make its total width less than 49 units. But glue is allowed to stretch arbitrarily far, whenever it has a positive stretch component.

►**Exercise 12.1:** How wide would the glue globs be if the horizontal list in the illustrations were to be made 100 units wide?

Ⓓ  $\TeX$  is somewhat reluctant to stretch glue more than its stated stretchability, as we shall see later when we discuss the "badness" of particular glue settings. Therefore if you are trying to decide how big to make each aspect of the glue in some layout, the rules are: (a) The natural glue space should be the amount of space that looks best. (b) The glue stretch should be the maximum amount of space that can be added to the natural spacing before the layout begins to look bad. (c) The glue shrink should be the maximum amount of space that can be subtracted from the natural spacing before the layout begins to look bad.

In most cases the designer of a book layout will have specified all the kinds of glue that are to be used, so a typist will not need to decide how big any glue attributes should be. For example, the *Art of Computer Programming* layout in Appendix E includes the definition of three control sequences `\xskip`, `\yskip`, and `\yyskip`. A typist for those books will insert `\xskip` within a paragraph in certain places where a little extra stretchability is appropriate; and `\yskip` is inserted between paragraphs when the paragraphs discuss somewhat different topics. Even more space is inserted before and after theorems and algorithms, etc.; this is called `\yyskip` because it is twice as much glue as `\yskip`. (The same three control sequences have been used when preparing this manual. For example, "`\xskip`" appears in the paragraph preceding this one, just before "(a)", "(b)", and "(c)"; and "`\yyskip`" is used before and after every "dangerous bend" paragraph like the next one.)

Ⓓ To specify glue in a horizontal list of boxes, without using a predefined format like `\xskip`, type "`\hskip(dimen) plus(dimen) minus(dimen)`". The "`plus(dimen)`" and "`minus(dimen)`" specify stretch and shrink components. They are optional; and if left out, the corresponding glue component has length zero. The space component, however, must always be given, even when it is zero; and if zero, you must remember to type "0pt", not just "0". If you are omitting the shrink component, the next characters of your text had better not be "minus". If you are omitting both stretch and shrink components, the next characters of your text had better not be "plus". Similar remarks apply to the specification of glue in vertical lists; the only difference is that you type "`\vskip`" instead of "`\hskip`".

There is one aspect of glue that a careful typist will want to be aware of, namely that  $\TeX$  automatically increases the stretchability (and decreases the

shrinkability) after punctuation marks. The reason for this is that it's usually better to put more space after a period than between two ordinary words, when spreading a line out to reach the desired margins. Consider, for example, the following sentences from a classic kindergarten pre-primer:

``Oh, oh!`` cried Baby Sally. Dick and Jane laughed.

If  $\text{\TeX}$  sets this at its natural width, all the spaces will be the same:

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

But if the line needs to be expanded by 5 points, 10 points, 15 points, or more,  $\text{\TeX}$  will set it as

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.


“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

and so on. There is no glue between adjacent letters, so individual words will always look the same. The glue after the comma stretches at 1.25 times the rate of the glue between adjacent words; the glue after the period and after the ! `` stretches at 3 times the rate. Furthermore if  $\text{\TeX}$  had to shrink this line to its minimum width, the result would be

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

The glue after a comma shrinks only 80 per cent as much as ordinary inter-word glue, and after a period or exclamation point it shrinks by only one third as much.

 The exact rule  $\text{\TeX}$  uses at a space is this: Each font tells  $\text{\TeX}$  what glue to use for spaces when that font is active. When starting to process a horizontal list,  $\text{\TeX}$  sets an internal variable called the “space factor” to 1. When appending a character to a horizontal list, the space factor is changed to 3 if the character is a period, question mark, or exclamation point (as determined by its ascii code); it is changed to 2 if the character is a colon, to 1.5 if a semicolon, to 1.25 if a comma. The space factor is left unchanged if the character being appended is a ) or ] or ' or " ; and it is reset to 1 whenever any other character or math formula or non-character box is appended. Furthermore, the space factor remains unchanged when appending a character immediately following an upper case letter. (The reason for this is to avoid treating the period specially when it merely follows an initial, like the periods in “P. A. M. Dirac”.) When a space is encountered, the glue space is taken from the current font glue space specification; the stretch and shrink are obtained by respectively multiplying and dividing the font glue stretch and shrink specifications by the space factor.

The only trouble with this rule is that it fails when a period isn't really a period ... like when it is used (as in this sentence) to make an "ellipsis" of three dots, or when it is used after abbreviations. If, for example, you are typing a bibliographic reference to *Proc. Amer. Math. Soc.*, you don't want the glue after these periods to be any different from the ordinary inter-word glue. The best way to handle this is to use "escape space" after a non-sentence-ending period, e.g., to type

```
Proc.\ Amer.\ Math.\ Soc.
```

This works because the space in "\ " always has the unmodified inter-word glue of the current font. Granted that this input looks a bit ugly, it does give the best-looking output. It's one of those things we occasionally have to do when dealing with a computer that tries to be smart.

◊ ▶ **Exercise 12.2:** How can you defeat the rule the other way, for sentences like "... launched by NASA.?"

Incidentally, if you try to specify "..." by typing three periods in a row, you get "..."—the dots are too close together. The best way to handle this is to go into *mathematics* mode, using the `\ldots` control sequence defined in *basic* T<sub>E</sub>X format. For example, if you type

```
Hmmm $\ldots$ I wonder why?
```

the result is "Hmmm ... I wonder why?" The reason this works is that math formulas are exempt from normal text spacing rules. Chapter 17 has more to say about `\ldots` and related topics.

One of the interesting things that happens when glue stretches and shrinks at different rates is that there might be glue with essentially *infinite* stretchability. For example, consider again the four boxes we had above, with the same glue as before except that the glue in the middle has stretchability 999997 (nearly one million) instead of 6. Now the total stretchability is one million; and when the line has to grow, almost all of the additional space will get put into the middle glue. If, for example, a box of width 58 is desired, the first glue expands from 9 to 9.000018 units, the middle glue from 9 to 14.999982 units, and of course the last glue remains exactly 12 units thick. For all practical purposes, the spacing has gone from 9, 9, 12 to 9, 15, 12.



If such infinitely stretchable glue is placed at the left of a row of boxes, the effect is to *right justify* them, i.e., to move them over to the rightmost boundary of the constructed box. And if you take two globs of infinitely stretchable glue, putting one at the left and one at the right, the effect is to *center* the list of boxes within a larger box. This in fact is how the `\ctrline` instruction works: it places infinite glue at both ends, then makes a box of width `\hsize`. [Actually the stretchability is 1000 cm, namely 10 meters (about 33 feet); that isn't infinite, but it's close enough.]

② The glue actually used in the definition of `\ctrline` is `\hskip Opt plus 1000cm minus 1000cm`; in other words, *both stretch and shrink components are essentially infinite*. The reason is that if you try to center something that is bigger than the actual `\hsize`, it will be centered but will extend into the margins; the glue at left and right will shrink from 0 to something *negative*. Like box dimensions, glue components can be negative, and this is occasionally useful for things like backspacing.

② "Infinite" glue can be specified in a horizontal list by typing "`\hf111`", or in a vertical list by typing "`\vf111`". An `\hf111` instruction is equivalent to `\hskip Opt plus 10000000000pt` (that's ten *billion* points), and `\vf111` is equivalent to `\vskip` by the same amounts. We have already seen a typical use of `\vf111` in the example of Chapter 6.

### <13> Modes

Just as people get into different moods, TeX gets into different "modes." (Except that TeX is more predictable than people.) There are six modes:

- Vertical mode. [Building the vertical list used to make the pages of output.]
- Restricted vertical mode. [Building a vertical list for a box within a page.]
- Horizontal mode. [Building the horizontal list used to make the next paragraph for the output pages.]
- Restricted horizontal mode. [Building a horizontal list for a box within a page.]
- Math mode. [Building a mathematical formula to be placed in a horizontal list.]

- Display math mode. [Building a mathematical formula to be placed on a line by itself, temporarily interrupting the current paragraph.]

In simple situations, you don't need to be aware of what mode  $\TeX$  is in, because it just does the right thing. But when you get an error message that says "You can't do that in horizontal mode", a knowledge of modes helps explain why  $\TeX$  thinks you goofed.

Basically  $\TeX$  is in one of the vertical modes when it is preparing a list of boxes and glue that will be placed vertically on top of one another; it's in one of the horizontal modes when it is preparing a list of boxes and glue that will be strung out horizontally next to each other with baselines aligned; and it's in one of the math modes when it is reading a math formula.

A play-by-play account of a typical  $\TeX$  job should make the mode idea clear: At the beginning,  $\TeX$  is in vertical mode, ready to construct pages. If you specify glue or a box when  $\TeX$  is in vertical mode, the glue or the box gets placed on the current page below what has already been specified. For example, the `\vskip` instructions in the sample run we discussed in Chapter 6 contributed vertical glue to the page; and the `\ctrlline{MY STORY}` instruction contributed a box to the page. While building the `\ctrlline` box,  $\TeX$  went temporarily into restricted horizontal mode, but returned to vertical mode after setting the glue in that box.

Continuing with the example of Chapter 6,  $\TeX$  switched into horizontal mode as soon as it read the "O" of "Once upon a time". Horizontal mode is the mode for making paragraphs. The entire paragraph up to the `\par` was input in horizontal mode; then it was divided into lines of the appropriate length, these lines were appended to the page (with appropriate glue between them), and  $\TeX$  was back in vertical mode.

In general when  $\TeX$  is in vertical mode, the first character of a new paragraph changes the mode to horizontal for the duration of a paragraph. If a begin-math character (\$) appears when in horizontal mode,  $\TeX$  plunges into math mode, processes the formula up until the closing \$, then adds the text of this formula to the current paragraph and returns to horizontal mode. (Thus, in the "I wonder why?" example of the previous chapter,  $\TeX$  would go into math mode temporarily while processing `\ldots`, treating the dots as a formula.)

However, if two consecutive begin-math characters appear in a paragraph (\$\$),  $\TeX$  interrupts the paragraph where it is, contributes the paragraph-so-far to the page, then processes a math formula in display math mode, then contributes


this formula to the current page, then returns to horizontal mode for more of the paragraph. (The formula to be displayed should end with `$$`.) For example, if you type

```
the number $$\pi \approx 3.1415926536$$ is important ,
```

`TeX` goes into display math mode between the `$$`'s, and the output you get states that the number

$$\pi \approx 3.1415926536$$

is important.

 `TeX` gets into restricted vertical mode when you ask it to construct a box from a vertical list of boxes (using `\vjust` or `\valign`) or when you do `\topinsert` or `\botinsert`. It gets into restricted horizontal mode when you ask it to construct a box from a horizontal list of boxes (using `\hjust` or `\halign`). Box construction is discussed in Chapter 21. Restricted modes are like the corresponding unrestricted ones except that you can't do certain things. For example, you can't say `$$` in restricted horizontal mode, because you're not making a paragraph. You can't begin a paragraph in restricted vertical mode, etc. All the rules about what you can do in various modes are summarized in Chapters 24–26.

When handling simple manuscripts, `TeX` spends almost all of its time in horizontal mode (making paragraphs), with brief excursions into vertical mode (between paragraphs).

At the end of a job, you type `\end` at some point when `TeX` is in vertical mode; this causes `TeX` to finish any unfinished pages and stop. (Actually it is better to type `\vfill\end` in most cases, since `\vfill` inserts enough space to fill up the last page properly. Without the `\vfill`, `TeX` attempts to stretch out the lines it has accumulated for the last page, with the bottom line appearing at the bottom of the page; you probably don't want this.)

#### <14> How `TeX` breaks paragraphs into lines

When the end of a paragraph is encountered, `TeX` determines the "best" way to break it into lines. In this respect, `TeX` gives better results than most other typesetting systems, which produce each separate line of output before beginning the next, because the *final* words of a `TeX` paragraph can influence how the lines

at the *beginning* are broken. T<sub>E</sub>X's new approach to this problem (based on "sophisticated computer science techniques"—whew!) requires only a little more computation than the traditional methods, and leads to significantly fewer cases when words need to be hyphenated.

T<sub>E</sub>X does try to hyphenate words, but it uses a hyphenation only when there is no better alternative. The complete rules by which T<sub>E</sub>X hyphenates words are given in Appendix H. They are sufficiently simple that you could memorize them and apply them by hand if you wanted to, but there probably isn't any need for you to know them in detail. Basically T<sub>E</sub>X's approach to hyphenation is one of *extreme caution*: instead of trying to find all legitimate places where a hyphen could occur, T<sub>E</sub>X sticks to hyphenations that appear to be quite safe.

In view of T<sub>E</sub>X's improved line-breaking methods, this cautious approach to hyphenation is usually satisfactory; but every once in a while, like all automatic approaches to language processing, it fails. The reason for failure is generally that a rather long nonstandard word has occurred: T<sub>E</sub>X refuses to apply automatic hyphenation to a sequence of boxes unless that sequence

- a) consists entirely of lower case letters belonging to a single font; and
- b) is preceded immediately by glue (e.g., a space); and
- c) is followed immediately by glue or by a punctuation mark (something that doesn't set the "space factor" to 1, cf. Chapter 12).

One consequence of these conditions is that proper names and words containing accented letters will not be hyphenated; but such words tend to disobey the normal hyphenation rules anyway. Another consequence is that T<sub>E</sub>X won't mess around with words for which you have explicitly prescribed the hyphenation. And already-hyphenated compound words won't be broken up any further.

In spite of these apparently severe restrictions, experience shows that T<sub>E</sub>X works amazingly well in practice, except when the margins are extremely close together (small `\hspace`); and *nothing* works very well in that case. (A large dictionary, combined with T<sub>E</sub>X's line-breaking method, would do the best conceivable job; but for normal books and journals it isn't worthwhile for the computer to waste time referring to a large dictionary. T<sub>E</sub>X's program and tables for hyphenation require only about 3000 words of computer memory, so they place little burden on the overall processing.) When proofreading the output of T<sub>E</sub>X, the amount of additional work needed to correct missed hyphenations is quite negligible compared to the amount of work that proofreading already involves.

When you do find a word that  $\TeX$  should have hyphenated but didn't, or when you find one of the extremely rare cases in which  $\TeX$  inserts a hyphen in the wrong place, the remedy is to revise the manuscript, telling  $\TeX$  how to hyphenate the offending word by inserting *discretionary hyphens*. The control sequence “\‑” indicates a discretionary hyphen, namely a place where a word may be hyphenated if there is no better alternative.


For example, if you run into a situation where the French word *mathématique* must be hyphenated, you can type it as

```
math\-\`e\‑ma\‑tique .
```

Another word  $\TeX$  has trouble with is “onomatopoeia”; if necessary, type it in as

```
on\‑o\‑mat\‑o\‑poeia .
```

(Or you could use the fancy “œ” ligature, cf. Chapter 9.) But don't bother to insert any discretionary hyphens until after  $\TeX$  has failed to find a good way to break lines in some paragraph.

 Before describing  $\TeX$ 's neat method for breaking a paragraph up into lines, we should discuss the rules for all legal breaks in a paragraph. Here they are: Outside of math formulas, you can break a paragraph

- a) at glue, provided that the glue is immediately preceded by a character box or a constructed box (but not a rule box), or by the end of a math formula, or by a discretionary hyphen, or by an *insertion* (`\topinsert` or `\botinsert`, which are explained in Chapter 15).
- b) where a `\penalty` has been specified in horizontal mode (see below), provided that the penalty is less than 1000.
- c) at a discretionary hyphenation (with the hyphen included in the text, taken from the font that was current at the time the `\‑` appeared), paying a penalty of 50.
- d) where `\eject` has been specified (see below—this is a way to end a page at a particular place within a paragraph).
- e) after “‑” or any ligature that ends with “‑” (thus, in standard roman fonts this means after “‑”, “--”, or “---”).

Inside math formulas, you can break

- a) after a binary operation like “+” (paying a penalty of 95), or after a relation like “=” (paying a penalty of 50).
- b) where a `\penalty` has been specified (see below), provided that the penalty is less than 1000.
- c) at a “discretionary math hyphen” specified by “\\*” (this inserts a multiplication sign  $\times$  into the formula), paying a penalty of 50.
- d) where `\eject` has been specified.

Note that some breaks are “free” but others have an associated penalty. Penalties are used to indicate the relative desirability of certain breaks. Breaks at `\eject` are compulsory; all other breaks are optional. When a break occurs at glue or just before glue, this glue disappears.

☞ TeX’s procedure for line breaking is based on the notion of the “badness” of glue setting. This is a technical concept defined by a formula that assigns a badness of 100 to a box in which glue had to stretch or shrink to its total amount of stretchability or shrinkability, while the badness is near zero if the glue’s stretchability or shrinkability is not very fully utilized. Furthermore the badness increases rapidly when glue is stretched to more than its stated limit; for example, the badness is 800 if the glue is stretched by twice its stretchability. Here is a precise way to calculate the badness, given that the total amount of glue stretch and shrink are  $y$  and  $z$ , respectively, and given that the box is supposed to grow by an amount  $x$  more than its natural width when the glue is set: Case 1,  $x \geq 0$  (stretching). If  $y < 10^{-4}$ , replace  $y$  by  $10^{-4}$ . Then the badness is  $100(x/y)^3$ . Case 2,  $x < 0$  (shrinking). If  $z < 10^{-4}$ , replace  $z$  by  $10^{-4}$ . Then the badness is  $100|x/z|^3$  if  $|x| \leq z$ , otherwise it is  $\infty$  (infinitely bad).

☞ When breaking lines of a paragraph, TeX essentially considers all ways to break the lines so that no line will have badness  $B$  exceeding 200. Such breaks are called “feasible.” Subject to this feasibility condition, TeX finds the best overall way to break, in the sense that the minimum total number of demerits occurs, where the demerits for each line of output are calculated as follows: If the penalty  $P$  for breaking at the end of this line is  $\geq 0$ , the number of demerits is  $(B + P + 1)^2$ ; if  $P < 0$ , the number is  $(B + 1)^2 - P^2$ . Furthermore an additional 3000 demerits are charged if two consecutive lines are being hyphenated or if the second-last line of the paragraph is hyphenated. A “dynamic programming” technique is used to find the breaks that lead to fewest total demerits. An attempt is made to hyphenate all words that meet the requirements mentioned earlier, whenever such words would straddle the end of line following some feasible break. The hyphenation algorithm of Appendix H is used to insert discretionary hyphens in all permissible places in such words. In practice the computation is quite fast, and only a few hyphenations need to be attempted, except in long paragraphs.

⊠ The current value of `\hspace` at the close of the paragraph is used to govern the width of each line, unless you specify “hanging” indentation. If you type “`\hangindent <dimen> for <number>`”, the specified dimension is supplied as an extra indentation on the first  $n$  lines of the paragraph, where  $n$  is the specified number. (That’s how the second line of the paragraph you’re reading was indented.) If you type “`\hangindent <dimen> after <number>`”, the specified dimension is supplied as an extra indentation on all but the first  $n$  lines of the paragraph. If you type just “`\hangindent<dimen>`”, then “after 1” is assumed. If the specified dimension is negative, indentation occurs at the right margin instead of at the left.

⊠ TeX indents the first line of each paragraph by inserting an empty box of width `\parindent` at the beginning, unless you start the paragraph by typing the control sequence `\noindent`.


⊠ The number 200 used to determine feasibility can be changed to  $100n$  for any integer  $n \geq 2$  by typing “`\jpar<number>`”, where  $n$  is the specified number. A large value of  $n$  will cause TeX to run more slowly, but it makes more line breaks feasible in cases where lines are so narrow that  $n = 2$  finds no solutions.


⊠ The instruction `\ragged<number>` specifies a degree of “raggedness” for the right-hand margins. If this number is  $r$ , the line width changes towards its natural width by the ratio  $r/(100 + r)$ . Thus, `\ragged 0` (the normal setting) gives no raggedness; `\ragged 100` causes the width of each line to be midway between `\hspace` and its natural width; and `\ragged 1000000` almost completely suppresses any stretching or shrinking of the glue. Some people like to use this “ragged right margin” feature in order to make the output look less formal, as if it hadn’t actually been typeset by an inhuman computer. (Some people also think that “ragged right” typesetting saves money. On traditional typesetting equipment, this was true, but computer typesetting has changed the situation completely: the most expensive part of the computation is now the breaking of lines, while the setting of glue costs almost nothing.)

⊠ The numbers 50, 3000, 95, and 50 used in the above rules for hyphenation penalties, consecutive-hyphenation demerits, binary-operation-break penalties, and relation-break penalties, can be changed by typing `\chpar2+<number>`, `\chpar3+<number>`, `\chpar6+<number>`, and `\chpar7+<number>`, respectively. Hyphenation penalties in force at the end of a paragraph are used throughout that paragraph; relation and operator penalties in force at the opening `$` of a math formula are used throughout that formula.

⊠ To insert a penalty at a specified point in a paragraph, simply type “`\penalty <number>`”. Any penalty  $\geq 1000$  is equivalent to a penalty of  $\infty$  (a non-permissible


place to break); any penalty  $< 1000$  implies that a break at the current place is permissible. The penalty may be zero or even negative, to indicate an especially desirable break location.

 The control sequence `\eject` forces a break at the position where `\eject` occurs, and also causes  $\TeX$  to begin the next line on a new page. This gives you a way to remake page 100, say, without changing page 101, provided that it is possible to end the new page 100 at the same place where page 101 begins. Note that `\eject` will make the last line of the paragraph-so-far reach to the right-hand margin (if feasible); this is what some printers call a "quad middle" operation. It is quite different from what you would get if you simply typed "`\par`" at the spot that the revised page should end.  $\TeX$ 's linebreaking algorithm is especially advantageous when handling `\eject`, because it has an apparent ability to "look ahead."

 Additional vertical glue specified by `\parekip` is inserted just before each paragraph. This glue gets added to the normal interline glue.

### <15> How $\TeX$ makes lists of lines into pages

$\TeX$  attempts to choose desirable places to stop making up one page and start another, and its technique for doing this usually works pretty well. But if you don't like the way a page is broken, you can force a page break in your favorite place by typing "`\eject`". An `\eject` command can occur in vertical mode (e.g., between paragraphs) or in horizontal mode (within a paragraph) or even in math mode; but you won't need to make much use of it.

  $\TeX$  groups things into pages in much the same way as it makes up paragraphs, except for the lookahead feature. Badness ratings and penalties are used to find the best place to break, but each page break is made once and for all when this "best" place is found—otherwise  $\TeX$  would have to remember the contents of so many pages, it would run out of memory space. Legal breaks between pages can occur

- a) at glue, provided that the glue is immediately preceded by a constructed box (but not a rule box). This includes the glue routinely inserted between lines, as explained below.
- b) where a `\penalty` has been specified in vertical mode, provided that the penalty is less than 1000. (Cf. Chapter 14.)
- c) after an insertion (arising from `\topinsert` or `\botinsert`, see below).
- d) where `\eject` is specified.

Breaks at `\eject` are compulsory; all other breaks are optional. When a break occurs at glue or just before glue, this glue disappears.



When boxes are appended to any vertical list (in particular, when they are appended to the current page), glue is automatically placed between them so that the distance between adjacent baselines tends to be the same. For example, the lines of 9-point text you are now reading have baselines 11 points apart. This implies that the glue between lines is not always the same, because more glue space is inserted under a line whose characters all stay above the baseline than under a line having characters that descend below it. Such interline glue is appended just before each box even when you have explicitly inserted glue yourself with `\vskip` or `\vfll`; any glue you specify is in addition to the interline glue.

Here is how interline glue gets figured: The book designer has specified two kinds of glue by using the operations `\baselineskip` (glue) and `\lineskip` (glue). Suppose the `\baselineskip` glue has  $x$  units of space,  $y$  units of stretch, and  $z$  units of shrink. (In this paragraph `TeX` is using  $x = 11$  points,  $y = z = 0$ , but  $y$  and  $z$  need not be zero.) Suppose we are appending a box of height  $h$  to a vertical list in which the previous box (ignoring glue) had depth  $d$ . Then the interline glue inserted just above the new box will have  $x - h - d$  units of space,  $y$  units of stretch, and  $z$  units of shrink, whenever  $x - h - d \geq 0$ ; but if  $x - h - d < 0$ , the interline glue will be the glue specified by `\lineskip`. For example, the basic `TeX` format in Appendix B says `"\baselineskip 12 pt \lineskip 1 pt"`; this means that baselines will normally be 12 points apart, but when this is impossible a space of 1 point will be inserted between adjacent boxes of a vertical list. *Exception:* Interline glue is not inserted before or after rule boxes, nor is it inserted before the first box or after the last box of a vertical list.

Contributions are made to the current page until the accumulated page height minus the accumulated glue shrinkability first exceeds the specified page size. (Page size is specified by the book designer using `\size`, see below.) At this point the break is made at whatever legal break in the page-so-far results in fewest badness-plus-penalty points  $B + P$ , where the badness  $B$  is defined as in Chapter 14 (except using vertical glue), and where the penalty  $P$  is zero unless explicitly specified or included by the paragraphing routine. The paragraphing routine inserts a penalty of 80 points just after the first line and just after the penultimate line of a multi-line paragraph, with an additional penalty of 50 points just after a line that ends with a hyphenation. This tends to avoid so-called "widows" (i.e., breaks that leave only one line of a paragraph on a page); for example, `TeX` breaks a four-line paragraph without 80 points of penalty only by breaking it into  $2 + 2$  lines. A penalty of 500 points is charged for breaking pages just before a displayed equation. Furthermore there is a penalty of 80 for breaking after the first line of text that follows a display, unless the paragraph ends with such a line. (There is no penalty for breaking before the last line of text that precedes a display, since such a line is not considered to be a "widow.") Once the best break

has been identified, the page is output, glue at the break is deleted, and everything remaining is contributed to the following page. (To change the numbers 80, 50, and 500 relating to widow-line, broken-line, and display-break penalties, you can use the `\chpar` instruction as explained in Chapter 24.)

② The height of a page is the value of `\vsize`, and the depth in most cases is the depth of the bottom line on that page. Thus, if one page has 10-point type and the next has 9-point type, the baselines at the bottoms of both pages will be at the same place even though the descenders of 10-point letters go slightly further below the baseline than the descenders of 9-point letters do. However, the bottom line on a page is sometimes a constructed box whose depth is very large, and in such a case we want the baseline to be higher. T<sub>E</sub>X deals with the problem as follows: Whenever a box having depth greater than `\maxdepth` is contributed to the current page (where "`\maxdepth{dimen}`" has been specified by the book designer), the depth of the page-so-far is artificially decreased to `\maxdepth`, and the height of the page-so-far is correspondingly increased. (Interline glue calculation is not affected by this artificial adjustment, except possibly afterwards when the page is being dealt with as a completed box.) There is also another design parameter, "`\topbaseline{dimen}`", which is used to insert glue at the top of the page so that the baseline of the first box will be at least this distance from the top (if it isn't a rule box). All other glue is normally deleted at the top of each page; to put glue there, simply insert a `\null` box first. If several different values of `\vsize`, `\maxdepth`, or `\topbaseline` occur in the same T<sub>E</sub>X job, each page is governed by the values in force when the first item was contributed to that page.

② A "floating-insertion" capability is built into T<sub>E</sub>X so that, among other things, illustrations can be placed at the top of the first subsequent page on which they fit, and footnotes can be placed at the bottom of the page on which the footnote reference appears. Here's how it works: You type "`\topinsert{<vlist>}`" or "`\botinsert{<vlist>}`", where `<vlist>` is a sequence of instructions that specifies a vertical list of boxes and glue. If such an insertion is made when T<sub>E</sub>X is in vertical mode, the specified vertical list will be contributed to the first page on which there is room for it. If such an insertion is made when T<sub>E</sub>X is in horizontal mode, the specified vertical list will be contributed to the same page on which the line containing the insertion appears. A `\topinsert` is contributed at the top, a `\botinsert` at the bottom. Glue specified by `\topskip{glue}` will be placed just below every `\topinsert`; glue specified by `\botskip{glue}` will be placed just above every `\botinsert`.

② You may be wondering how things like page numbers get attached to pages. Actually T<sub>E</sub>X has two levels of control: when a complete page has been built, this page is packaged as a box and another section of T<sub>E</sub>X input code comes into action. The designer has specified this other piece of code by writing "`\output{...}`", and we will

discuss the details of `\output` routines in Chapter 23. For now, it should suffice to give just a small taste of what an `\output` routine looks like:

```
\output{\baselineskip 20pt
        \page\ctrlline{:a\count0}\advcount0}
```

This routine (which appears in Appendix B) takes the current page number, typeset in font `a`, and centers it on a new line below the contents of the current page; “`\page`” means the current page, “`\count0`” means the current page number, and “`\advcount0`” advances this number by 1. The baseline of the page number will be 20 points below the baseline of the page—assuming that `\maxdepth` has been set small enough that this is always possible. This setting of `\baselineskip` will be retracted at the end of the `\output` routine, according to the normal conventions of grouping; thus there will be no effect on `TeX`'s page-building operations (which go on asynchronously).

### <16> Typing math formulas

`TeX` was designed to handle complex mathematical formulas in such a way that most of them are easy to input. The basic idea is that a complicated formula is composed of less complicated formulas put together in a simple way, and these less complicated formulas are in turn made up of simple combinations of formulas that are even less complicated, and so on. Stating this another way, if you know how to type simple formulas and how to combine formulas into larger ones, you will be able to handle virtually any formula at all. So let's start with simple ones and work our way up.

The simplest formula is a single letter, like “ $x$ ”, or a single number, like “2”. In order to enter these into a `TeX` text, you type “ $\$x\$$ ” and “ $\$2\$$ ”, respectively. Note that all mathematical formulas are enclosed in special math brackets, and we are using `$` as the math bracket in this manual, in accord with the basic `TeX` format defined in Appendix B. Note further that when you type “ $\$x\$$ ” the “ $x$ ” comes out in italic type, but when you type “ $\$2\$$ ” the “2” comes out normally. In general, all characters on your keyboard have a special interpretation in math formulas, according to the normal conventions of mathematics printing. Letters now denote italic letters, while digits and punctuation denote roman digits and punctuation; a hyphen (`-`) now denotes a minus sign (`—`), which is almost the same as an em-dash but not quite (see Chapter 2). So if you forget one `$` or type one `$` too many, `TeX` will probably become thoroughly confused and you will probably get some sort of error message.

Formulas that have been typeset by a printer who is unaccustomed to doing mathematics usually look quite wrong to a mathematician, because a novice printer usually gets the spacing all wrong. In order to alleviate this problem,  $\text{\TeX}$  does most of its own spacing in math formulas; and it ignores any spaces you type between  $\$$ 's. For example, you can type  $\$ x\$$  and  $\$ 2 \$$  and they will mean the same thing as  $\$x\$$  and  $\$2\$$ ; you can type  $\$(x + y)/(x - y)\$$  or  $\$(x+y) / (x-y)\$$ , but both will result in  $(x + y)/(x - y)$ . Thus, you are free to use blank spaces in any way you like. Of course, spaces are still used in the normal way to mark the end of control sequences, as explained in Chapter 7. In most circumstances  $\text{\TeX}$ 's spacing will be what a mathematician is accustomed to; but we will see in Chapter 18 that there are control sequences by which you can override  $\text{\TeX}$ 's spacing rules if you want.

One of the things mathematicians like to do is make their formulas look like Greek to the uninitiated. In  $\text{\TeX}$  language you can type  $\$\$ \backslash alpha, \backslash beta, \backslash gamma, \backslash delta; \$\$$  and you will get the first four Greek letters

$$\alpha, \beta, \gamma, \delta;$$

furthermore there are upper case Greek letters like  $\Gamma$ , which you can get by typing either  $\$\backslash Gamma\$$  or  $\$\backslash GAMMA\$$ . A few of the Greek letters deserve special attention: For example, lower case epsilon ( $\epsilon$ ) is quite different from the symbol used to denote membership in a set ( $\in$ ); type  $\$\backslash epsilon\$$  for  $\epsilon$  and  $\$\backslash in\$$  for  $\in$ . Furthermore, three of the lower case Greek letters have variant forms on  $\text{\TeX}$ 's standard italic fonts;  $\$(\backslash phi, \backslash theta, \backslash omega)\$$  yields  $(\phi, \theta, \omega)$  while  $\$(\backslash varphi, \backslash vartheta, \backslash varomega)\$$  yields  $(\varphi, \vartheta, \varpi)$ .

Besides Greek letters, there are a lot of funny symbols like  $\approx$  (which you get by typing  $\$\backslash approx\$$ ) and  $\mapsto$  (which you get by typing  $\$\backslash mapsto\$$ ). A complete list of these control sequences and the characters they correspond to appears in Appendix F. The list even includes some non-mathematical symbols like

$$\S \ \dagger \ \ddagger \ \S \ \copyright \ \text{\pounds}$$

which you can get by typing  $\$\backslash section\$$ ,  $\$\backslash dag\$$ ,  $\$\backslash ddag\$$ ,  $\$\backslash PS\$$ ,  $\$\backslash copyright\$$ ,  $\$\backslash \$\$$ , and  $\$\backslash sterling\$$ , respectively; nearly all of the special symbols that you'll ever want are available in this way. Such control sequences are allowed only in math mode, i.e., between  $\$$ 's, even when the corresponding symbols aren't traditionally considered to be mathematical, because they appear in the math fonts.

Now let's see how more complex formulas get built up from simple ones. In the first place, you can get superscripts and subscripts by using "↑" and "↓":

Type	and you get
<code>\$x↑2\$</code>	$x^2$
<code>\$x↓2\$</code>	$x_2$
<code>\$2↑x\$</code>	$2^x$
<code>\$x↑2y↑2\$</code>	$x^2y^2$
<code>\$x ↑ 2y ↑ 2\$</code>	$x^2y^2$
<code>\$x↓2y↓2\$</code>	$x_2y_2$
<code>\$↓2F↓3\$</code>	${}_2F_3$

Note that ↑ and ↓ apply only to the next single character. If you want several things to be subscripted or superscripted, just enclose them in braces:

<code>\$x↑{2y}\$</code>	$x^{2y}$
<code>\$2↑{2↑x}\$</code>	$2^{2^x}$
<code>\$2↑{2↑{2↑x}}\$</code>	$2^{2^{2^x}}$
<code>\$x↓{y↓2}\$</code>	$x_{y_2}$
<code>\$x↓{y↑2}\$</code>	$x_{y^2}$

It is illegal to type "x↑y↑z" or "x↓y↓z" (TeX will complain of a "double superscript" or "double subscript"); you must type "x↑{y↑z}" or "{x↑y}↑z" or "x↑{yz}" in order to make your intention clear. (Some commonly-used languages for math typesetting treat x↑y↑z as x↑{y↑z} and others treat it as {x↑y}↑z or x↑{yz}; the ambiguous construction isn't needed much anyway, so TeX disallows it.)

A subscript or superscript following nothing (as in the "↓2F↓3" example above, where the ↓2 follows nothing) is taken to mean a subscript or superscript of an empty box. A subscript or superscript following a character applies to that character only, but when following a box it applies to that whole box; for example,

<code>\$((x↑2)↑3)↑4\$</code>	$((x^2)^3)^4$
<code>\$\$\{(\{x↑2\})↑3\}↑4\$</code>	$((x^2)^3)^4$

In the first formula the ↑3 and ↑4 are superscripts on the right parentheses, but in the second formula they are superscripts on the formulas enclosed in braces.

You can have simultaneous subscripts and superscripts, and you can specify them in any order:

<code>\$x↑2↓3\$</code>	$x_3^2$
<code>\$x↓3↑2\$</code>	$x_3^2$
<code>\$x↑{31415}↓{92}+\pi\$</code>	$x_{92}^{31415} + \pi$
<code>\$x↓{y↑a↓b}↑{z↓c↑d}\$</code>	$x_{y_b}^{z_c^d}$

Note that simultaneous sub/superscripts are positioned over each other, aligned at the left.

The control sequence `\prime` stands for the character "′", which is used mostly in superscripts. Here's a typical example:

<code>\$y↓1↑\prime+y↓2↑{\prime\prime\prime}\$</code>	$y_1 + y_2'''$
--	----------------

Another way to get complex formulas from simple ones is to use the control sequences `\sqrt`, `\underline`, or `\overline`. These operations apply to the character or group that follows them:

<code>\$\$\sqrt{2}\$</code>	$\sqrt{2}$
<code>\$\$\sqrt{x+2}\$</code>	$\sqrt{x+2}$
<code>\$\$\underline{4}\$</code>	$\underline{4}$
<code>\$\$\underline{\underline{4}}\$</code>	$\underline{\underline{4}}$
<code>\$x↑{\underline{n}}\$</code>	$x^{\underline{n}}$
<code>\$\$\overline{x↑3+\sqrt{3}}\$</code>	$\overline{x^3 + \sqrt{3}}$

❖ If you need cube roots (or  $n^{\text{th}}$  roots), TeX has no built-in mechanism for this. But you can insert a 3 (or  $n$ ) over a square root sign by using Appendix B's control sequence `\spos` for superposition. Type

`\spos{\raise{dimen}\hjust{\hskip{dimen}$\scriptscriptstyle{root}$}}`

followed by `\sqrt{...}`, where you can figure out appropriate dimensions by fiddling around until the position looks right. (These dimensions depend on the size of the formula, the current size of type, and the size of the square root sign.) For example, " $\sqrt[3]{5}$ " can be set with TeX's normal 10-point fonts by typing

`$$\spos{\raise5pt\hjust{\hskip2.5pt$\scriptscriptstyle3$}}\sqrt5$`

Ⓛ Accents in math mode work something like `\overline`; you can accent a single character or a formula. (But the formula had better be short, since a tiny accent will be centered over the whole thing.) For example,

`\=x+\overline x+\b x+\A x+\s x+\s{\s x}+\A(x+y)+e†{\=x}`

produces  $\bar{x} + \bar{x} + \hat{x} + \hat{x} + \tilde{x} + \tilde{x} + x \dagger y + e^{\ddagger}$ .

►Exercise 16.1: What would you type to get the following formulas?

$2^{n+1}$      $(n+1)^2$      $\sqrt{1-x^2}$      $\overline{w+z}$      $p_1^{e_1}$      $a_{b_c d_e}$      $f''_n(x)$

►Exercise 16.2: What's wrong with typing the following?

If  $x = y$ , then  $x$  is equal to  $y$ .

►Exercise 16.3: Explain how to type the following sentence:

Deleting an element from an  $n$ -tuple leaves an  $(n-1)$ -tuple.

### <17> More about math

Another thing mathematicians like to do is make fractions—and they also like to build up symbols on top of each other, as in

$\frac{1}{2}$     and     $\frac{n+1}{3}$     and     $\begin{bmatrix} n+1 \\ 3 \end{bmatrix}$     and     $\sum_{n=1}^3 Z_n$  .

You can get these four formulas by typing "`\over 2`" and "`n+1\over 3`" and "`n+1\comb[] 3`" and "`\sum_{n=1}^3 Z_n`"; we shall study the simple rules for such constructions in this chapter.

First let's look at fractions, which use the "`\over`" notation. The control sequence `\over` applies to everything in the formula unless you enclose `\over`

in a { } group; in the latter case it applies to everything in that group.

Type	and you get
<code>\$\$x+y↑2\over k+1\$\$</code>	$\frac{x+y^2}{k+1}$
<code>\$\$x+(y↑2\over k)+1\$\$</code>	$x+\frac{y^2}{k}+1$
<code>\$\$x+(y↑2\over k+1)\$\$</code>	$x+\frac{y^2}{k+1}$
<code>\$\$x+y↑{2\over k+1}\$\$</code>	$x+y^{\frac{2}{k+1}}$

You aren't allowed to use `\over` twice in the same group; instead of typing a formula like "a `\over` b `\over` 2", you must specify what goes over what:

<code>\$\$\{a\over b\}\over 2\$\$</code>	$\frac{\frac{a}{b}}{2}$
<code>\$\$a\over\{b\over 2\}\$\$</code>	$\frac{a}{\frac{b}{2}}$

Note that the letters get smaller when they are fractions-within-fractions, just as they get smaller when they are used as exponents. It's about time that we studied how  $\TeX$  does this. Actually  $\TeX$  has eight different styles in which it can treat formulas, namely

display style	(for formulas displayed on lines by themselves)
text style	(for formulas embedded in the text)
script style	(for formulas used as superscripts or subscripts)
scriptscript style	(for second-order superscripts or subscripts)

and four other styles that are almost the same except that exponents aren't raised quite so much. For brevity we shall refer to the eight styles as

*D, T, S, SS, D', T', S', SS'*,



so that  $T$  is text style,  $D'$  is modified display style, etc.  $\text{\TeX}$  also uses three sizes of type for mathematics, called text size, script size, and scriptscript size ( $t$ ,  $s$ , and  $ss$ ).

The normal way to typeset a formula with  $\text{\TeX}$  is to enclose it in dollar signs  $\$ \dots \$$ , which yields the formula in text style (style  $T$ ), or to enclose it in double dollar signs  $\$ \$ \dots \$ \$$ , which displays the formula in display style (style  $D$ ). Once you know the style, you can determine the size of type  $\text{\TeX}$  will use:

If a letter is in style	then it will be set in size
$D, T, D', T'$	$t$
$S, S'$	$s$
$SS, SS'$	$ss$

There is no " $SSS$ " style or " $sss$ " size; such tiny symbols would be even less readable than the  $ss$  ones. Therefore  $\text{\TeX}$  stays with  $ss$  as its minimum size, as shown in the following chart:

In a formula of style	the superscript style is	and the subscript style is
$D, T$	$S$	$S'$
$S, SS$	$SS$	$SS'$
$D', T'$	$S'$	$S'$
$S', SS'$	$SS'$	$SS'$

For example, if  $x \uparrow \{a \downarrow b\}$  is in style  $D$ , then  $\{a \downarrow b\}$  is in style  $S$ , and  $b$  is in style  $SS'$ .

So far we haven't seen any difference between styles  $D$  and  $T$ . Actually there is a slight difference in the positioning of exponents: you get  $x^2$  in  $D$  style and  $x^2$  in  $T$  style and  $x^2$  in  $D'$  or  $T'$  style—do you see the difference? But there is a big distinction between  $D$  style and  $T$  style when it comes to fractions:

In a formula $\alpha \text{\backslash over} \beta$ of style	the style of the numerator $\alpha$ is	and the style of the denominator $\beta$ is
$D$	$T$	$T'$
$T$	$S$	$S'$
$S, SS$	$SS$	$SS'$
$D'$	$T'$	$T'$
$T'$	$S'$	$S'$
$S', SS'$	$SS'$	$SS'$

Thus if you type "`$1\over2$`" (in a text) you get  $\frac{1}{2}$ , namely style *S* over style *S'*; but if you type "`$$1\over2$$`" you get

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

(a displayed formula), which is style *T* over style *T'*.

When a fraction like `$x+y\over z$` is put into the text of a paragraph, the letters are rather small and hard to read:  $\frac{x+y}{z}$ . So it is usually better to type the fraction in the mathematically equivalent way "`$(x+y)/z$`", which comes out " $(x+y)/z$ ". In other words, `\over` is useful mostly for displayed formulas or for numeric fractions.

While we're at it, we might as well finish the style rules: `\underline` does not change the style; `\sqrt` and `\overline` both change *D* to *D'*, *T* to *T'*, *S* to *S'*, *SS* to *SS'*, and leave *D'*, *T'*, *S'*, *SS'* unchanged.

There's another operation "`\atop`", which is like `\over` except that it leaves out the fraction line:

$$\text{\code{$$x\atop y+2$$}} \quad \begin{array}{c} x \\ y+2 \end{array}$$

The basic math definitions in Appendix B also define "`\choose`", which is like `\atop` but it encloses the result in parentheses:

$$\text{\code{$$n\choose k$$}} \quad \binom{n}{k}$$

This is a common notation for the so-called "binomial coefficient" that tells how many ways there are to choose *k* things out of *n* things; that's why the control sequence is called `\choose`.

You can't mix `\over` and `\atop` and `\choose` with each other. For example, "`$$n \choose k \over 2$$`" is illegal; you must use grouping, to get either "`$$\{n \choose k\} \over 2$$`" or "`$$n \choose \{k \over 2\}$$`", i.e.,

$$\frac{\binom{n}{k}}{2} \quad \text{or} \quad \binom{n}{\frac{k}{2}} .$$

The latter formula, incidentally, would look better as “`$$n \choose k/2$$`” or “`$$n \choose {1\over 2}k$$`”, yielding

$$\binom{n}{k/2} \quad \text{or} \quad \binom{n}{\frac{1}{2}k} .$$

Suppose you don't like the style TeX selects by its automatic style rules. Then you can specify the style you want by typing

`\dispstyle` or `\textstyle` or `\scriptstyle` or `\scriptscriptstyle`.

For example, if you want the  $\binom{n}{k}$  to be larger in the formula `$$\{n\choose k\}\over 2$$`, just type “`$$\dispstyle{n\choose k}\over 2$$`”; you will get

$$\frac{\binom{n}{k}}{2}$$

because the numerator of the formula is now “`\dispstyle{n\choose k}`”. Here's another example (admittedly a rather silly one): `$$n+\scriptstyle n+\scriptscriptstyle n$$` gives

$$n + n + n .$$

Note that the plus signs get smaller too, as the style changes; and there's no space around `+` signs in script style.

►Exercise 17.1: Explain how to specify the displayed formula

$$\binom{p}{2} x^2 y^{p-2} - \frac{1}{1-x} \frac{1}{1-x^2} .$$

Ⓔ There are two other variants of `\over`, `\atop`, etc. First is “`\above<dimen>`”, which is just like `\over` but the stated dimension specifies the exact thickness of the line rule. For example,

$$$$\dispstyle{x\over y}\above 1pt\dispstyle{w\over z}$$$$

will produce

$$\frac{\frac{z}{y}}{\frac{w}{z}}$$

this sort of thing was once customary in arithmetic textbooks, but nowadays it is rare (at least in pure mathematics). The second variant is a generalization of `\choose`: You can write "`\comb{<delim>}{<delim>}`", specifying any of the delimiters listed in Chapter 18; "`\choose`" is the same as "`\comb{}`", and one of the examples at the beginning of this section used "`\comb{}`".

⊠ When you use `\over`, `\atop`, etc., the numerator and denominator are centered over each other. If you prefer to have the numerator or denominator at the left, follow it by "`\hfill`"; if you prefer to have it at the right, precede it by "`\hfill`". For example, the specification

```
$$1+{\1\hfill\over\displaystyle a_1+{\1\hfill\over\displaystyle
a_2+{\1\hfill\over\displaystyle a_3+{\1\over a_4}}}}$$
```

yields

$$1 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4}}}}$$

while without the `\hfills` you get

$$1 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4}}}}$$

Mathematicians often use the sign  $\sum$  to stand for "summation" and the sign  $\int$  to stand for "integration." If you're a typist but not a mathematician, all you need to remember is that `\sum` stands for  $\sum$  and `\int` for  $\int$ ; these abbreviations appear in Appendix F together with all the other symbols, in case you forget.

Symbols like  $\sum$  and  $\int$  (and a few others like  $\cup$  and  $\prod$  and  $\oint$  and  $\otimes$ , all listed in Appendix F) are called *large operators*, and you type them just as you type ordinary symbols or letters. The difference is that TeX will choose a *larger* large operator in display style than it will in text style. For example,

$$\begin{array}{ll} \text{\texttt{\$}\sum x\downarrow n\text{\texttt{\$}}} & \text{yields } \sum x_n \quad (T \text{ style}) \\ \text{\texttt{\$\$\sum x\downarrow n\$\$\text{\texttt{\$}}} & \text{yields } \sum x_n \quad (D \text{ style}). \end{array}$$

Usually  $\sum$  occurs with "limits," i.e., with formulas that are to appear below it or to the right. You type limits just the same as superscripts and subscripts: for example, if you want


$$\sum_{n=1}^m$$

you type either "`\sum\downarrow\{n=1\}\uparrow m`" or "`\sum\uparrow m\downarrow\{n=1\}`". According to the normal conventions of mathematics, TeX will change this to " $\sum_{n=1}^m$ " if in text style rather than display style.

Integrations are slightly different from summations, in that the limits get set at the right even in display style:

$$\begin{array}{ll} \text{\texttt{\$}\int\downarrow\{-\infty\}\uparrow\{+\infty\}\text{\texttt{\$}}} & \text{yields } \int_{-\infty}^{+\infty} \quad (T \text{ style}) \\ \text{\texttt{\$\$\int\downarrow\{-\infty\}\uparrow\{+\infty\}\$\$\text{\texttt{\$}}} & \text{yields } \int_{-\infty}^{+\infty} \quad (D \text{ style}). \end{array}$$

Note further that the subscript is not directly below the superscript, in either style; again, this is a mathematical convention that TeX follows automatically (based on information stored with the fonts).

 Some printers prefer to set limits above and below  $\int$  signs; similarly, some prefer to set limits to the right of  $\sum$  signs. You can change TeX's convention by simply typing "`\limitswitch`" after the large operator. For example,

$$\begin{array}{ll} \text{\texttt{\$\$\int\limitswitch\downarrow\{-\infty\}\uparrow\{+\infty\}\$\$\text{\texttt{\$}}} & \text{yields } \int_{-\infty}^{+\infty} \\ \text{\texttt{\$\$\sum\limitswitch\downarrow\{n=1\}\uparrow m\$\$\text{\texttt{\$}}} & \text{yields } \sum_{n=1}^m \end{array}$$

Ⓢ If you have to put two or more rows of limits under a large operator, you can do this by using “\atop”. For example, if you want the displayed formula

$$\sum_{\substack{0 \leq i \leq m \\ 0 < j < n}} P(i, j)$$

the correct way to type it is

`$$\sum_{\scriptstyle 0 \leq i \leq m \atop \scriptstyle 0 < j < n} P(i, j)$$`

(perhaps with a few more spaces to make it look nicer in the manuscript file). Note that the instruction “\scriptstyle” was necessary here, twice—otherwise “ $0 \leq i \leq m$ ” and “ $0 < j < n$ ” would have been in scriptscript size, which is too small. This is one of the rare cases where TeX's automatic style rules need to be overruled.

►Exercise 17.2: How would you type the displayed formula  $\sum_{i=1}^p \sum_{j=1}^q \sum_{k=1}^r a_{ij} b_{jk} c_{ki}$  ?



►Exercise 17.3: And how about  $\sum_{\substack{1 \leq i \leq p \\ 1 \leq j \leq q \\ 1 \leq k \leq r}} a_{ij} b_{jk} c_{ki}$  ?

## <18> Fine points of mathematics typing

We have discussed most of the facilities needed to construct math formulas, but there are several more things a good mathematical typist will want to watch for.

**1. Punctuation.** When a formula is followed by a period, comma, semicolon, colon, question mark, exclamation point, etc., put the punctuation after the \$, when the formula is in the text; but put the punctuation before the \$\$ when the formula is displayed. For example,

If  $\$x < 0\$$ , we have shown that  $$$y = f(x) .$$$

The reason is that TeX's spacing rules within paragraphs work best when the punctuation marks are not considered part of the formulas.

Similarly, don't type something like this:

for  $x = a, b$, or  $c$.$$

It should be

for  $x = a$,  $b$, or  $c$.$$$

The reason is that  $\TeX$  will always put a "thin space" between the comma and the  $b$  in  $x = a, b$.$  This space will probably not be the same as the space  $\TeX$  puts after the comma after the  $b$ , since the second comma is outside the formula; and such unequal spacing would look bad. When you type it right, the spacing will look good. Another reason for not typing " $x = a, b$$ " is that it inhibits the possibilities for breaking lines in a paragraph:  $\TeX$  will never break at the space between the comma and the  $b$  because breaks after commas in formulas are usually wrong. For example, in the equation " $x = f(a, b)$$ " we certainly don't want to put " $x = f(a,$ " on one line and " $b)$$ " on the next.

Thus, when typing formulas in the text of a paragraph, keep the math properly segregated: Don't take operators like  $-$  and  $=$  outside of the  $\$$ 's, and keep commas inside the formula if they are truly part of the formula. But if a comma or period or other punctuation mark belongs linguistically to the sentence rather than to the formula, leave it outside the  $\$$ 's.

**2. Roman letters in formulas.** The names of algebraic variables in formulas are usually italic or Greek letters, but common mathematical operators like "log" are always set in roman type. The best way to deal with such operators is to make use of the following control sequences defined in the basic format of Appendix B:

<code>\cos</code>	<code>\exp</code>	<code>\lim</code>	<code>\log</code>	<code>\sec</code>
<code>\cot</code>	<code>\gcd</code>	<code>\liminf</code>	<code>\max</code>	<code>\sin</code>
<code>\csc</code>	<code>\inf</code>	<code>\limsup</code>	<code>\min</code>	<code>\sup</code>
<code>\det</code>	<code>\lg</code>	<code>\ln</code>	<code>\Pr</code>	<code>\tan</code>

The following examples show that such control sequences lead to roman type as

desired:

Type	and you get
<code>\sin2\theta=2\sin\theta\cos\theta</code>	$\sin 2\theta = 2 \sin \theta \cos \theta$
<code>O(n\log n\log\log n)</code>	$O(n \log n \log \log n)$
<code>\exp(-x^2)</code>	$\exp(-x^2)$
<code>\max_{1 \leq n \leq m} \log_2 P_n</code>	$\max_{1 \leq n \leq m} \log_2 P_n$
<code>\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1</code>	$\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$

In the second example, note that  $O$  is an upper case letter “oh”, not a zero; a formula should usually have “0” instead of “O” when a left parenthesis follows. The fourth and fifth examples show that some of the special control sequences are treated by TeX as “large operators” with limits just like  $\sum$ ; compare the different treatment of subscripts applied to  $\max$  and to  $\log$ .

Another way to get roman type into mathematical formulas is to include constructed boxes (cf. Chapter 21); such boxes are treated the same as single characters or subformulas. For example,

`\exp(x+\hjust{constant})` yields  $\exp(x + \text{constant})$ .

The fonts used inside such boxes are the same as the fonts used *outside* of the math brackets  $\dots$ ; the characters do *not* change size when the style changes.

►Exercise 18.1: Explain how to type the phrase “ $n^{\text{th}}$  root”, where “ $n^{\text{th}}$ ” is treated as a mathematical formula with a superscript. The letters “th” should be in font d.

There is, of course, a way to specify characters that do change size with changing styles; you can do it with the `\char` command. We studied `\char` in Chapter 8, but `\char` works a little differently in math mode because math mode deals with up to ten fonts instead of just one font. TeX keeps three fonts for text size, three for script size, and three for scriptscript size, plus one font for oversize and variable-size characters. The three fonts of changing size are called *rm*, *it*, and *sy* fonts—short for roman, italic, and symbols, according to TeX’s normal way of using these fonts; and the oversize font is called the *ex* font. (The *rm* and *it* fonts are essentially normal fonts like all other fonts TeX deals with, but each *sy* and *ex* font must have special control



information stored with it, telling T<sub>E</sub>X how to do proper spacing of math formulas. Thus, T<sub>E</sub>X is able to do math typesetting on virtually any style of font, provided that the font designer includes these parameters.) To specify which fonts you are using for mathematics, you type

```
\mathrm {font}<font><font>
\mathit {font}<font><font>
\mathsy {font}<font><font>
\mathex {font}
```

before getting into math mode, where the *rm*, *it*, and *sy* fonts are specified in the order text size, script size, scriptscript size. For example, by typing “\mathit tpk” you are saying that T<sub>E</sub>X should use font *t* as the *it* font in text size math, font *p* as the *it* font in script size math, font *k* in scriptscript size math. If you don't use scriptscript size in your formulas, you must still specify a font, but you could say “\mathit tpp” or even “\mathit ttt”. (When you specify a font letter for the first time you must follow it with the font file name, as described in Chapter 4; e.g., “\mathit t+cmi10 p+cmi7 p” would work. But it's best to declare all your fonts first, before specifying the ones to be used for math.) Now... about that “\char” operation in math mode: Although \char selects up to 128 characters in non-math modes, it selects up to 512 characters in math mode. Characters '000 to '177 are in the *rm* font of the current size, '200 to '377 are in the *it* font of the current size, '400 to '577 are in the *sy* font of the current size, and '600 to '777 are in the *ex* font. For example, the “dangerous bend” road symbol is in the *ex* font being used to typeset this user manual, and it is actually character number '177 in this font, so it is referred to by typing “\char'177”. The symbol ∞ is character number '61 in T<sub>E</sub>X's standard symbol fonts; in math mode you can refer to it either as “\infty” or as “\char'461”, or simply as “∞” if you happen to have this key on your keyboard.

Ⓢ T<sub>E</sub>X fonts used for variables (“*it*” fonts) have spacing appropriate for math formulas but not for italic text. You should use a different font for “italicized words” in the text. For example:

*This sentence is in font cmi10, which is intended for formulas, not text.*

*This sentence is in font cmti10, which is intended for text, not formulas.*

**3. Large parentheses and other delimiters.** Since mathematical formulas can get horribly large, T<sub>E</sub>X has to have some way to make ever-larger symbols. For example, if you type

```
$$\sqrt{1+\sqrt{1+\sqrt{1+
\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}$$
```

the result shows a variety of available square-root signs:

$$\sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + x}}}}}}}$$

The three largest signs here are all essentially the same, except for a vertical segment " | " that gets repeated as often as necessary to reach the desired size; but the smaller signs are distinct characters found in TeX's math fonts.

A similar thing happens with parentheses and other so-called "delimiter" symbols. For example, here are the different sizes of parentheses that TeX might use in formulas:



The three largest pairs are made with repeatable extensions, so they can become as large as necessary.

TeX chooses the correct size of square root sign by simply using the smallest size that will enclose the formula being `\sqrted`, but it does not use large parentheses or other delimiters unless you ask it to. If you want to enclose a formula in variable-size delimiters, type

$$\backslashleft\langle\text{delim}_1\rangle\langle\text{formula}\rangle\backslashright\rangle\langle\text{delim}_2\rangle$$

where each `\langle delim \rangle` is one of the following:

<code>.</code>	blank ( )	<code> </code>	vertical line (   )
<code>(</code>	left parenthesis ( ( )	<code>\ </code>	double vertical line (    )
<code>)</code>	right parenthesis ( ) )	<code>\langle</code> or <code>&lt;</code>	left angle bracket ( < )
<code>[</code>	left bracket ( [ )	<code>\rangle</code> or <code>&gt;</code>	right angle bracket ( > )
<code>]</code>	right bracket ( ] )	<code>\lfloor</code>	left floor bracket ( ⌊ )
<code>\{</code>	left brace ( { )	<code>\rfloor</code>	right floor bracket ( ⌋ )
<code>\}</code>	right brace ( } )	<code>\lceil</code>	left ceiling bracket ( ⌈ )
<code>/</code>	slash ( / )	<code>\rceil</code>	right ceiling bracket ( ⌉ )

For example, if you type `$$1+ \left( 1\over1-x^2 \right) ^3$$` you will get

$$1 + \left( \frac{1}{1-x^2} \right)^3 .$$

Notice from this example that `\left` and `\right` have the effect of grouping just as `{` and `}` do: The `\over` operation does not apply to the `"1+"` or to the `"^3"`, and the `"^3"` applies to the entire formula enclosed by `\left(` and `\right)`.

When you use `\left` and `\right` they must match each other, nesting like braces do in groups. You can't have `\left` in one formula and `\right` in another, nor can you type things like `"\left(...\{...\right)..."`. This restriction makes sense, of course, but it is worth explicit mention here because you do *not* have to match parentheses and brackets, etc., when you are not using `\left` and `\right`: `TeX` will not complain if you input a formula like `"$[0,1)$"` or even `"$)($"`. (And it's a good thing `TeX` doesn't, for such unbalanced formulas occur surprisingly often in mathematics papers.) Even when you are using `\left` and `\right`, `TeX` doesn't look closely at the particular delimiters you happen to choose; thus, you can type strange things like `"\left)"` and/or `"\right("` if you know what you're doing. Or even if you don't.

If you type `"\left."` or `"\right."`, the corresponding delimiter is blank—not there. Why on earth would anybody want that, you may ask. Well, there are at least two reasons. One is to take care of situations like this:

$$|x| = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{if } x < 0. \end{cases}$$

The formula in this case could be typed as follows:

```
$$|x|=\left\{ ... \right.$
```

where `"..."` stands for a `TeX` box containing the text

$x,$	if $x \geq 0;$
$-x,$	if $x < 0.$

Later in this chapter we shall discuss how you might specify such a box; just now we are simply trying to discuss the use of a blank delimiter.

The second example of a blank delimiter occurs when you want a variable-size slash; type either "`\left/ ... \right.`" or "`\left. ... \right/`", whichever will make the correct size slash (i.e., a slash that is just big enough for the formula enclosed between `\left` and `\right`). For example, if you want to get the formula

$$\frac{a+1}{b} / \frac{c+1}{d}$$

you can type either "`$$\left. a+1 \over b \right/ {c+1\over d}$$`" or "`$$\{a+1\over b\} \left/ c+1 \over d \right.$`".

⊠ A third example, which occurs less often, is the problem of getting three large delimiters of the same size, as in a formula of the form "`[ a | β ]`" where  $a$  and  $\beta$  are large formulas and, say,  $a$  is bigger than  $\beta$ . You can type

`\left.\left[ a \right| \beta \right]`

to handle this. Note that a construction like "`\left(\left( ... \right)\right)`" will always produce double parentheses of the same size.

The size chosen by  $\TeX$  when you use `\left` and `\right` is usually appropriate, but there is an important exception: When the `\left` and `\right` enclose a displayed  $\sum$  or  $\prod$ , etc., with upper and/or lower limits,  $\TeX$  will often make the delimiters much too large. For example, if you type

`$$\left( \sum_{i=1}^n A_i \right)^2$$`

the result is

$$\left( \sum_{i=1}^n A_i \right)^2$$

(rather shocking). The reason is that  $\TeX$  adds extra blank space above and below the limits so that they don't interfere with surrounding formulas; usually this is the right thing to do, except when large delimiters are involved. In fact, most math compositors prefer to let the limits on  $\sum$ 's protrude above or below any enclosing parentheses, so `\left` and `\right` aren't really the proper things to type anyway. What you should do is use control sequences such as `\bigglp` and

`\biggrp`, which are defined in the basic  $\TeX$  format (Appendix B). When the above example is retyped in the form

```
$$\bigg\lrcorner \sum_{i=1}^n A_i \bigg\rrcorner 2$$
```

it will come out right:

$$\left(\sum_{i=1}^n A_i\right)^2 .$$

Incidentally, basic format also defines two other useful sizes of parentheses, for those occasions when you wish to control the size by yourself in a convenient manner: `\biglp` and `\bigrp` produce parentheses that are just a little bit bigger than normal ones, while `\biggglp` and `\bigggrrp` produce really big ones. Here is a typical example of a formula that uses `\biglp` and `\bigrp`:

$$(x - s(x))(y - s(y)).$$


►**Exercise 18.2:** Explain exactly how to type this formula so that  $\TeX$  would typeset it as shown.

◊ Instead of using “big” delimiters, there is another way to get  $\TeX$  to choose a more reasonable size with respect to displayed  $\sum$ 's with limits, namely to fool  $\TeX$  into thinking that the formulas aren't as big as they really are. Using Appendix B, type “`\chop to <dimen>{(formula)}`” to produce a box containing the specified formula in display style but with the depth of the box artificially assumed to be the specified dimension. The `<dimen>` must be in points (pt). For example,

```
\sqrt{\chop to 9pt{\sum_{1\le k\le n} A_k}}
```

yields

$$\sqrt{\sum_{1\le k\le n} A_k} .$$

 You can also access other delimiters that might be present in your fonts by using the versatile `\char` command. We saw above that `\char` has an extended meaning in math mode; its meaning is even further extended when used to specify delimiters. Besides the options listed above, any  $\langle\text{delim}\rangle$  can be “`\char`c1c2`” where  $c_1$  and  $c_2$  are three-digit octal codes;  $c_1$  is the code for this delimiter in its smaller sizes (rm, *it*, or *sy* fonts) and  $c_2$  is the code for this delimiter in the *ex* font. For example, it turns out that the left brace delimiter can be specified as `\char`546610`, since a normal size left brace is character ``146` in the *sy* font, and since all oversize left braces are reachable starting at character ``010` in the *ex* font. (Characters in an *ex* font are internally linked together in order of increasing size.) You should let  $c_1$  or  $c_2$  equal 000 if there is no corresponding character.  $\text{\TeX}$  handles variable-size delimiters in the following way: If  $c_1 \neq 000$ , the first step is to look at math character ``c1` in the current size, then in any larger sizes. (For example, in script style  $\text{\TeX}$  looks first at script size character ``c1`, then at the corresponding character in text size.) If  $c_2 \neq 000$ , the next step is to look at all characters linked together in the *ex* font, starting at ``c2`, in increasing order of size. (This linked list might end with an extensible character.) The first character  $\text{\TeX}$  sees that is large enough (i.e.,  $\geq$  the desired size) is chosen. Special note to those who have read this far: Standard *ex* fonts for  $\text{\TeX}$  often contain the “left pretzel” and “right pretzel” delimiters that you can get by typing

`\left\char`000656`    and    `\right\char`000657,`

respectively. Startle your friends by using these instead of parentheses around your big matrices, or try typing “`$$\left\char`656\quad\text{\vcenter{\hjust to 250pt{\dots several sentences of text \dots}}\quad\right\char`657}$$`”.

**4. Spacing.** Chapter 16 says that  $\text{\TeX}$  does automatic spacing of math formulas so that they look right, and this is almost true, but occasionally you must give  $\text{\TeX}$  some help. The number of possible math formulas is vast, and  $\text{\TeX}$ 's spacing rules are rather simple, so it is natural that exceptions should arise. Furthermore there are occasions when you need to specify the proper spacing between two formulas. Perhaps the most common example of this is a display containing a main formula and side conditions, like

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2.$$

You need to tell  $\text{\TeX}$  how much space to put after the comma.

The traditional hot-metal technology for printing has led to some ingrained standards for situations like this, based on what printers call a “quad” of space. Since these standards seem to work well in practice,  $\text{\TeX}$  makes it easy for you to continue the tradition. When you type “ $\backslash\text{quad}$ ”,  $\text{\TeX}$  converts this into an amount of space equal to a printer’s quad, approximately the width of a capital M. (The em-dash discussed in Chapter 2 is usually one quad wide; and one quad in 10-point type is usually equal to 10 points. This is where the name “quad” comes from; it once meant a square piece of blank type. But of course a font designer is free to specify any sizes that he or she wants for the widths of quads, em-dashes, and M’s.)

The abbreviation “ $\backslash\text{qqquad}$ ” is defined in Appendix B to be the same as “ $\backslash\text{quad}\backslash\text{quad}$ ”, and this is the normal spacing for situations like the  $F_n$  example above. Thus, the recommended procedure is to type

$$F_n = F_{n-1} + F_{n-2}, \backslash\text{qqquad } n \geq 2.$$

It is perhaps worth reiterating that  $\text{\TeX}$  ignores all the spaces in math mode (except, of course, the space after “ $\backslash\text{qqquad}$ ”, which is needed to distinguish “ $\backslash\text{qqquad } n$ ” from “ $\backslash\text{qqquad}n$ ”); so the same result would be obtained if you were to type

$$F_n = F_{n-1} + F_{n-2}, \backslash\text{qqquad } n \geq 2.$$

Thus, all spacing that differs from the normal conventions has to be specified explicitly by control sequences such as  $\backslash\text{quad}$  and  $\backslash\text{qqquad}$ .

Of course,  $\backslash\text{quad}$  and  $\backslash\text{qqquad}$  are big chunks of space, more than the space between words in a sentence, so it is desirable to have much finer units. The basic elements of space that  $\text{\TeX}$  deals with in math formulas are often called a “thin space” and a “thick space”, defined respectively to be  $\frac{1}{8}$  of a quad and  $\frac{5}{18}$  of a quad. In order to get a feeling for these units, let’s take a look at the  $F_n$  example again: thick spaces occur just before and after the  $=$  sign, and also before and after the  $\geq$  sign. A thin space is slightly smaller, yet quite noticeable; it’s a thin space that makes the difference between “loglog” and “log log”.

$\text{\TeX}$  has variable glue, as we discussed in Chapter 12, so spaces in  $\text{\TeX}$ ’s math formulas actually can get a little thicker or thinner when a line is being stretched or squeezed. Here is a precise chart telling about all the different kinds of spaces that you can specify in math formulas:

Control sequence	Name	Spacing in styles $D, T, D', T'$	Spacing in styles $S, SS, S', SS'$
<code>\,</code>	Thin space	( 1/8, 0, 0)	( 1/8, 0, 0)
<code>\l</code>	Control space	( 2/9, 1/9, 2/9)	( 1/6, 0, 0)
<code>\&gt;</code>	Op space	( 2/9, 1/9, 2/9)	( 0, 0, 0)
<code>\;</code>	Thick space	( 5/18, 5/18, 0)	( 0, 0, 0)
<code>\quad</code>	Quad space	( 1, 0, 0)	( 1, 0, 0)
<code>\z</code>	Conditional thin space	( 1/6, 0, 0)	( 0, 0, 0)
<code>\!</code>	Negative thin space	( -1/8, 0, 0)	( -1/8, 0, 0)
<code>\?</code>	Negative thick space	( -5/18, -5/18, 0)	( 0, 0, 0)
<code>\&lt;</code>	Negative op space	( -2/9, -1/9, -2/9)	( 0, 0, 0)
<code>\s</code>	Negative <code>\z</code>	( -1/6, 0, 0)	( 0, 0, 0)

(Don't try to memorize this chart, just plan to use it for reference in case of need.) The spacing is given in units of quads; thus, for example, the entry "(5/18, 5/18, 0)" for a thick space in  $D$  style means that a thick space in displayed formulas is  $\frac{5}{18}$  of a quad wide, with a stretchability of  $\frac{5}{18}$  quad and a shrinkability of zero. Note that spacing is different in subscript or superscript styles: thick spaces disappear while thin spaces stay the same. This reflects the fact that no space surrounds = signs in subscripts, but there still remains a space in "log log" when you type "`\log\log`" in a script style.

The control sequences in this table are allowed only in math mode, except that `\quad` is allowed also in horizontal mode. Actually `\l` and `\!` are used in horizontal mode too, but with a different meaning explained earlier. It is permissible to use `\hskip` explicitly in math mode, if you want to specify any nonstandard glue.

As mentioned earlier, you will probably not be using any of these spaces very much. You can probably get by with only an occasional `\quad` (or `\qqquad`) and an occasional thin space.

In fact, there are probably only three occasions on which you should always remember to insert a thin space ("`\,`"):

- a) Before the  $dx$  or  $dy$  or  $d$  whatever in formulas involving calculus. For example, type "`\int\limits_0^\infty e^x dx`" to get " $\int_0^\infty e^x dx$ "; type "`dx dy = r dr d\theta`" to get " $dx dy = r dr d\theta$ ". (But type "`dy/dx`".)
- b) After square roots that happen to come too close to the following symbol. For example, "`0\biglp 1/\sqrt n\bigrp`" comes out as " $O(1/\sqrt{n})$ ",



but `"$O\biglp 1/\sqrt n\,\biggrp$"` yields  $O(1/\sqrt{n})$ . And it sometimes looks better to put a thin space after a square root to separate it visually from a symbol that follows:  $\sqrt{2}x$  is preferable to  $\sqrt{2}x$ , so type `"$\sqrt2\,x$"` instead of `"$\sqrt2 x$"`.

- c) After an exclamation point (which stands for the "factorial" operation in a formula) when it is followed by a letter or number or left delimiter. For example, `"$(2n)!\over n!\,(n+1)!"$`.

Other than this, you can usually rely on TeX's spacing until after you look at what comes out, and it shouldn't be necessary to insert optical spacing corrections except in rather rare circumstances. (One of these circumstances is a formula like  $\log n(\log \log n)^2$ , where a thin space has been inserted just before the left parenthesis; TeX inserts no space before this parenthesis, because similar formulas like  $\log f(x)$  want no space there. Another case is a formula like  $n/\log n$ , where a negative thin space has been inserted after the slash.)

Here are the rules TeX uses to govern spacing: The styles and sizes of all portions of a formula are determined as explained in Chapter 17. We may assume that the formula doesn't have the form  $a\over b$  (or  $a\atop b$ , etc.), since numerators and denominators of such formulas are treated separately. We may also assume that all subformulas of a given formula have been processed already (using the same rules) and replaced by boxes. (Subformulas include anything enclosed in  $\{ \dots \}$ , possibly combined with `\sqrt`, `\underline`, `\overline`, or `\accent`; subformulas also include anything enclosed in `\left<delim1> \dots \right<delim2>`, unless this turns out to be the entire formula. Subscripts and superscripts are attached to the appropriate boxes, and so any given formula can be reduced to a list of boxes to be placed next to each other; all that remains is to insert the appropriate spacing. The boxes are divided into seven categories:

- Ord box; e.g., an ordinary variable like  $x$ , or a subformula like `\sqrt{x+y}` that has already been converted into a box.
- Op box; e.g., a  $\sum$  sign (together with its limits, if any), or an operator like `\log` that has already been converted into a box.
- Bin box; e.g., a binary operator like  $+$  or  $-$  or `\times` (but not  $/$ , which is treated as "Ord").
- Rel box; e.g., an  $=$  sign or a  $<$  sign or  $\neq$ .
- Open box; e.g., a left parenthesis or `\left<delim>`.
- Close box; e.g., a right parenthesis or `\right<delim>`.

• Punct box; a comma or semicolon (but not a period, which is treated as "Ord").  
 Every Bin box must be preceded by an Ord box or a Close box, and followed by an Ord or Op or Open box, otherwise it is reclassified as Ord. (For example, in " $-\infty < x + y < +\infty$ ", only the + of " $x + y$ " is a Bin box; the two < signs are Rel boxes, and all other symbols are Ord boxes.) Now the spacing between any pair of adjacent boxes is determined by the following table:

		Right box type						
		Ord	Op	Bin	Rel	Open	Close	Punct
Left box type	Ord	0	\,	\>	\;	0	0	0
	Op	\,	\,	*	\;	0	0	0
	Bin	\>	\>	*	*	\>	*	*
	Rel	\;	\;	*	0	\;	0	0
	Open	0	0	*	0	0	0	0
	Close	0	\,	\>	\;	0	0	0
	Punct	\>	\>	*	\;	\>	\>	\>

Here "0" means no space is inserted; "\", " is a thin space; and so on. Table entries marked "\*" are never needed, because of the definition of Bin boxes.

For example, consider the displayed formula

$$x + y = \max\{x, y\} + \min\{x, y\}$$

which is transformed into the sequence of boxes

$$x \oplus y = \max \{ x, y \} \oplus \min \{ x, y \}$$

of respective types

Ord, Bin, Ord, Rel, Op, Open, Ord, Punct, Ord, Close, Bin, Op, Open, Ord, Punct, Ord, Close.

Inserting the appropriate spaces according to the table gives

$$\text{Ord} \backslash \text{Bin} \backslash \text{Ord} \ ; \ \text{Rel} \ ; \ \text{Op} \ \text{Open} \ \text{Ord} \ \text{Punct} \ \backslash \text{Ord} \ \text{Close} \\ \backslash \text{Bin} \ \backslash \text{Op} \ \text{Open} \ \text{Ord} \ \text{Punct} \ \backslash \text{Ord} \ \text{Close}$$


and the resulting formula is

$$x \oplus y = \max\{x, y\} \oplus \min\{x, y\}$$

i.e.,

$$x + y = \max\{x, y\} + \min\{x, y\}$$

This example doesn't involve subscripts or superscripts; but subscripts and superscripts merely get attached to boxes without changing the type of box. If you have inserted any spacing yourself by means of `\quad` or `\,` or `\hskip` or whatever,  $\TeX$ 's automatic spacing gets included in addition to what you specified. Similarly, if you have included `\penalty` or `\eject` or `\*` in a math formula, this specification is ignored for purposes of calculating the automatic glue between components of formulas. For example, if you type `"$. . . =\penalty100 x . . . $"`, there is a Rel box (`=`) followed by a penalty specification (which tends to avoid breaking lines here) followed by an Ord box (`x`), so  $\TeX$  inserts `"\;"` glue between the penalty and the Ord box.

 You can make  $\TeX$  think that a character or formula is Op or Bin or `\dots` or Punct by typing one of the instructions `\mathop(atom)` or `\mathbin(atom)` or `\mathrel(atom)` or `\mathopen(atom)` or `\mathclose(atom)` or `\mathpunct(atom)`, where `(atom)` is either a single character (like `x`), or a control sequence denoting a mathematics character (like `\gamma` or `\approx`), or `"\char(number)"`, or `"{\(formula)}`". For example, `"\mathopen|"` denotes a vertical line (absolute value bracket) treated as an Open box; and

$$\backslash\mathop{\backslash\char'155\backslash\char'141\backslash\char'170}$$

stands for the roman letters "max" in a size that varies with the math style. Control sequences like `\mathop` are used mostly in definitions of other control sequences for common idioms; for example, `"\max"` is defined in Appendix B to be precisely the above sequence of symbols. Note that there's no special control sequence to make a box "ordinary"; you get an Ord box simply by enclosing a formula in braces. For example, if you type `"{+}"` in a formula, the plus sign will be treated as an ordinary character like `x` for purposes of spacing. Another way to get the effect of `"{+}"` is to type `"\char'53"`, since characters entered with `\char` are considered ordinary.

**5. Line breaking.** When you have formulas in a paragraph,  $\TeX$  may have to break them between lines; it's something like hyphenation, a necessary evil that is avoided unless the alternative is worse. Generally  $\TeX$  will break a formula after a relation symbol like `=` or `<` or `\leftarrow`, or after a binary operation symbol like `+` or `-` or `\times`, if these are on the "outer level" of the formula (not enclosed in `{. . .}` and not part of an `"\over"` construction). For example, if you type

$$f(x,y) = x^2 - y^2 = (x+y)(x-y)$$

in mid-paragraph, there's a chance that  $\TeX$  will break after either of the `=` signs (it prefers this) or after the `-` or `+` or `\times` (in an emergency). Note that there won't

be a break after the comma in any case—commas after which breaks are desirable shouldn't ever appear between '\$'s. If you don't want to permit breaking in this example except after the = signs, you could type

$$\$f(x,y) = \{x^2-y^2\} = \{(x+y)(x-y)\}\$.$$

But it isn't necessary to bother worrying about such things unless  $\TeX$  actually does break a formula badly, since the chances of this are pretty slim.

⚠ There's a "discretionary hyphen" allowed in formulas, but it means multiplication: If you type " $\$(x+y)\*(x-y)\$$ ",  $\TeX$  will treat the  $\*$  something like the way it treats  $\-$ ; namely, a line break will be allowed at that place, with the hyphenation penalty. However, instead of inserting a hyphen,  $\TeX$  will insert a  $\times$  sign in the current size.

⚠ The penalty for breaking after a Rel box is 50, and the penalty for breaking after a Bin box is 95. These penalties can be changed either by typing " $\backslashpenalty\langle\text{number}\rangle$ " immediately after the box in question (thus changing the penalty in a particular case) or by using  $\backslashchpar$  as explained in Chapter 14 (thus changing the penalties applied at all subsequent Rel and/or Bin boxes of math formulas enclosed in the current group).

**6. Ellipses ("three dots").** Mathematical copy looks much nicer if you are careful about how "three dots" are typed in formulas and text. Although it looks fine to type "... " on a typewriter with fixed spacing, the result looks too crowded when you're using a printer's fonts:

$$"\$x \dots y\$" \quad \text{results in} \quad "x\dots y",$$

and such close spacing is undesirable except in subscripts or superscripts.

Furthermore there are two kinds of dots that can be used, one higher than the other; the best mathematical traditions distinguish between these. It is generally correct to produce formulas like

$$x_1 + \cdots + x_n \quad \text{and} \quad (x_1, \dots, x_n),$$

but wrong to produce formulas like

$$x_1 + \dots + x_n \quad \text{and} \quad (x_1, \dots, x_n).$$

When using TeX with the basic control sequences in Appendix B, you can solve the “three dots” problem in a simple way, and everyone will be envious of the beautiful formulas you produce. There are five main control sequences:

<code>\ldots</code>	three low dots ( ... );
<code>\cdots</code>	three center dots ( ... );
<code>\ldotss</code>	three low dots followed by a thin space;
<code>\cdotss</code>	three center dots followed by a thin space;
<code>\ldotsm</code>	three low dots preceded and followed by thin spaces.

Of these, “`\cdots`” and “`\ldotss`” are the most commonly used, as we shall see.

In general, it is best to use center dots between  $+$  and  $-$  signs, and also between  $=$  signs or  $\leq$  signs or  $\leftarrow$  signs or other similar relational operations. Lower dots are used between commas and when things are juxtaposed with no signs at all. Here are the recommended rules for using the above control sequences:

- a) Use `\cdots` between signs inside of a formula; use `\cdotss` just before punctuation at the end of a formula. Examples: “ $x_{1}=\cdots x_{n}=0$ ”; “the infinite sum  $y_{1}+y_{2}+\cdotss$ ”. (The extra thin space in `\cdotss` will make the second example look better than if `\cdots` had simply been used.)
- b) Use `\ldotss` before commas. Example:

The vector  $(x_{1}, \ldotss, x_{n})$  is composed  
of the components  $x_{1}$ ,  $\ldotss$ ,  $x_{n}$ .

This example deserves careful study. Note that the commas in the “vector” are part of the formula, but in the list of the components they are part of the sentence. Note also that you must be in math mode when using `\ldotss`.

- c) Use `\ldotsm` in “multiplicative” contexts, i.e., when three dots are used with no surrounding operator sign. Examples:

$x_{1}x_{2}\ldotsm x_{n}$ ;      $(1-x)(1-x^{2})\ldotsm(1-x^{k})$ .

Exception: Type “ $x^{1}x^{2}\ldotss x^{n}$ ”, because this formula when typeset ( $x^1x^2\dots x^n$ ) already has a “hole” at the baseline after  $x^2$ .

- d) Use `\ldots` in those comparatively rare cases where you want three lower dots without a thin space before or after them. Example: `"$(\ldots)$"`.
- e) Use `\cdots` between integral signs. Example:

$$\int \cdots \int f(x_1, \dots, x_n) dx_1 \dots dx_n.$$

- f) Use `"$\ldots\,$"` when a sentence ends with three lower dots. Example: "The periodic sequence 0, 1, 0, 1, 0, 1, `\ldots\,`"

**7. Handling vertical lines.** Besides the "idioms" represented by `\cdots` and `\ldots`, there are a few other situations that can be typeset more beautifully with a little care. A vertical line `|` and a double vertical line `||` are used for several different purposes in math formulas, and `TEX` will sometimes do a better job if you tell it what kind of a vertical line is meant. The following control sequences will help you in this task:

<code>\leftv</code>	vertical line used as a left parenthesis;
<code>\rightv</code>	vertical line used as a right parenthesis;
<code>\relv</code>	vertical line used as a relation.

For example, `"$\leftv +x \rightv = \leftv -x \rightv$"` specifies the displayed equation

$$|+x| = |-x| .$$

If this equation had been typed `"$|+x|=|-x|$"` the spacing would have been quite wrong, namely

$$|+x| = |-x| ,$$

because the `|`'s get the same spacing as ordinary variables like `x` when you haven't specified them to be `\leftv` or `\rightv` or `\relv`. Compare also the following two formulas:

<code>\$a b\$</code>	<code>a b ;</code>
<code>\$a\relv b\$</code>	<code>a b .</code>

There are three more control sequences `\leftvv`, `\rightvv`, and `\relvv`, which do the same for double vertical lines.

Appendix B defines two control sequences of use when specifying formulas like

$$\{x \mid x \geq 5\} .$$

The best way to type this is “`$$\leftset x \relv x \geq 5 \rightset$$`”, because `\leftset` and `\rightset` introduce braces with spacing to match the spaces surrounding the `\relv`.

**8. Number theory.** To specify a formula like “ $x \equiv y + 1 \pmod{p^2}$ ”, type “`$x \equiv y + 1 \pmod{p^2}$`”, using the control sequences `\equiv` and `\mod` defined in Appendix B. Note that you don't type the parentheses in this case; the control sequence provides them for you, with proper spacing and line-breaking conventions. (There is also a control sequence “`\neqv`” that produces the inequivalence symbol “ $\not\equiv$ ”.) To specify the formula

$$\gcd(m, n) = \gcd(n \bmod m, m) ,$$

type “`$$\gcd(m, n) = \gcd(n \bmod m, m)$$`”, using the control sequences `\gcd` and `\modop`. (Actually this latter formula would look slightly better if “`\,`” were inserted after the second comma.)

**9. Matrices.** Now comes the fun part. Many different kinds of matrices are used in mathematics, and you can handle them in `TeX` by using the general alignment procedures we shall be studying in a later chapter. For now, let's consider only simple cases. Suppose you want to specify the formula

$$A = \begin{pmatrix} x - \lambda & 1 & 0 \\ 0 & x - \lambda & 1 \\ 0 & 0 & x - \lambda \end{pmatrix} ;$$

here's how to do it:

```


$$A = \left( \vcenter{
    \halign{#\ctr{#}$\quad
            @#\ctr{#}$\quad
            @#\ctr{#}$\cr
    x - \lambda @1 @0 \cr
    0 @ x - \lambda @1 \cr
    0 @ 0 @ x - \lambda \cr
} } \right) $$


```

Explanation: We already know about “\left(” and “\right)”, which make the big parentheses that go around the matrix. The \vcenter control sequence forms a box in restricted vertical mode, and centers that box vertically so that the middle of the box is the same height as a minus sign. The \halign control sequence is one of the things you can do in restricted vertical mode; it is a general operator for producing aligned tables. After “\halign{” and up to the first “\cr” is a mysterious ritual for specifying three columns of a matrix. (We will learn the rules of this later, let's take it on faith just now.) Then comes a specification of the three matrix rows, with tab marks “@” between columns, and with pseudo-carriage-returns “\cr” at the end of each row. (Here @ is one of the special characters mentioned in Chapter 8, it is not the (tab) key on your keyboard; similarly, \cr is a control sequence, it is not (carriage-return). Furthermore \cr need not come at the end of a line; you can type several rows of a matrix on a single line of your TeX input manuscript.) After the final \cr comes the “)” to end “\halign{”; then comes the “)” to end “\vcenter{”. Finally the “\right)” finishes off the formula.

If there were five columns instead of three, the \halign specification would be about the same, only longer; namely,

```
\halign{${\ctr{#}}$\quad
        @${\ctr{#}}$\quad
        @${\ctr{#}}$\quad
        @${\ctr{#}}$\quad
        @${\ctr{#}}$\cr
```

followed by the individual rows. Here \ctr means that the corresponding column is to be centered; if you change it to \lft or \rt, the entries in the corresponding column will be set flush left or flush right, if they have different widths. When all matrix entries are numbers, it is usually better to use \rt than \ctr.

The \quads in the \halign ritual are used to specify the space between columns. If you want twice as much space you can replace \quad by \qqquad.

⊠ Another way to specify the matrix equation in the above example is to use the \cpile control sequence of Appendix B for each column:

```
$$A=\left(\cpile{x-\lambda\cr 0\cr 0\cr}\quad
          \cpile{1\cr x-\lambda\cr 0\cr}\quad
          \cpile{0\cr 1\cr x-\lambda\cr}\right)$$
```



However, this use of `\cpile` is *not* recommended, because it doesn't work in general: Each column is being typeset independently as a separate `\cpile`, so the rows won't line up properly if some matrix entries are taller than others. It's best to use `\halign` as suggested above—those funny-looking column format specifications are scary only the first few times you encounter them; afterwards they are quite simple to use. On the other hand `\cpile` (and its cousins `\lpile` and `\rpile`, which produce left-justified and right-justified columns of formulas just as `\cpile` produces centered columns) can be handy in simple cases.

❓ How about matrices involving `\ldots`? The following example should help you answer this question. Suppose you want to specify the matrix

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

One way to do it, using the "`\vdots`" control sequence of Appendix B, is

```


$$\begin{aligned} & \left( \begin{array}{cccc} \text{\center\halign{\ctr{#}}\; \!} \\ & \text{\ctr{#}}\; \! ; \text{\ctr{#}}\; \! ; \text{\ctr{#}}\; \! \\ a_{11} @ a_{12} @ \dots @ a_{1n} \backslash cr \\ a_{21} @ a_{22} @ \dots @ a_{2n} \backslash cr \\ \vdots @ \vdots @ \quad @ \vdots \backslash cr \\ a_{m1} @ a_{m2} @ \dots @ a_{mn} \backslash cr \end{array} \right) \end{aligned}$$


```

❓ Long ago in this chapter you were promised a solution to the problem of typing a displayed equation such as

$$|x| = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{if } x < 0. \end{cases}$$

Here it is, using `\vcenter` and `\halign`; see if you can understand it now:

```


$$\left( \begin{array}{l} x \right) = \left( \begin{array}{l} \text{\center\halign{\lft{##},\quad} \\ \text{\if \lft{##} \\ x \ge 0; \backslash cr} \\ -x < 0. \backslash cr} \end{array} \right)$$


```

Note that the commas and ifs are generated by the `\halign` specification; this trick isn't necessary, but it saves some typing. Another solution could be devised using `\lpile`, but (as in the discussion of matrices above) it is not recommended.

►Exercise 18.3: Explain how to type

$$\frac{1}{2\pi} \int_{-\infty}^{\sqrt{y}} \left( \sum_{k=1}^n \sin^2 x_k(t) \right) (f(t) + g(t)) dt.$$

►Exercise 18.4: Also explain how to type

$$\frac{(n_1 + n_2 + \dots + n_m)!}{n_1! n_2! \dots n_m!} = \binom{n_1 + n_2}{n_2} \binom{n_1 + n_2 + n_3}{n_3} \dots \binom{n_1 + n_2 + \dots + n_m}{n_m}.$$

◈►Exercise 18.5: How can you get T<sub>E</sub>X to typeset the column vector  $\begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}$  ?

◈►Exercise 18.6: Using Appendix F to find the names of special characters, explain how to type

$$\mathcal{P}_{Lh_j}(x) = \text{Tr} \left[ \frac{\partial F_{L-1}}{\partial t_h} \chi(L) \mathcal{M}_{n_j}(x) \right], \quad \text{evaluated at } \chi(\Gamma) \bmod SL(n, \mathbb{C}).$$

◈►Exercise 18.7: Define a control sequence `\e` for the “colon-equal” operator in computer science, so that a formula like “ $x := 2 \times x + 1$ ” will be properly spaced after it has been typed “`$x\e2\times x+1$`”.

### <19> Displayed equations

By now you know how to type mathematical formulas so that T<sub>E</sub>X will handle them with supreme elegance; but there is one more aspect of the art of mathematical typing that we should discuss. Namely, displays.

As mentioned earlier, you can type “`$$\langle formula \rangle$$`” to display a formula in flamboyant display style. Another thing you can do is type

$$\text{code: } \text{code} \backslash \text{eqno} \langle \text{formula} \rangle \text{code} ;$$

this displays the first formula and also puts an equation number (the second formula) at the right-hand margin. For example,

$$\text{code: } x^2 - y^2 = (x+y)(x-y) . \backslash \text{eqno} (15) \text{code}$$

will produce this:

$$x^2 - y^2 = (x + y)(x - y). \tag{15}$$

Here's what `$$` and `\eqno` do, in more detail: The formula to be displayed is made into a box using `display style` (unless you override the style). If `\eqno` appears, the formula following it is made into a box using `text style`. When the combined width of these two boxes, plus one text size quad, exceeds the current line width, squeezing is attempted as follows: If the shrinkability of the formula to be displayed would allow it to fit, the formula is repackaged into a box that has just enough width; otherwise the formula is repackaged into a box having the current line width, and the equation number (if any) will be placed on a new line just below the formula box. The formula to be displayed is centered on the line, where this centering is independent of the width of the equation number, *unless* this would leave less space between the formula and the equation number than the width of the equation number itself; in the latter case, the formula is placed flush left on the line. Now `TeX` looks at the length of the previous line of the current paragraph: if this is short compared to the size of the displayed equation, vertical glue that the designer has specified by `"\dispaskip{glue}"` will be placed above the formula, and vertical glue specified by `"\dispbskip{glue}"` will be placed below. Otherwise vertical glue specified by `"\dispskip{glue}"` will be placed both above and below. (The glue below is, however, omitted if an equation number has to be dropped down to a separate line; this separate line takes the place of the glue that ordinarily would have appeared.)

Another thing you can type for displays, when you know what you're doing, is `"$$\halign(spec){(alignment)}$$"`. This is just like an ordinary `\halign`, except that the `$$`'s interrupt a paragraph and insert `\dispskip` glue above and below the aligned result. Note that `\eqno` cannot be used in this case, and no automatic centering is done. Page breaks might occur in the midst of such displays.

OK, the use of displayed formulas is very nice, but when you try typing a lot of manuscripts you will run into some displays that don't fit the simple pattern of a single formula with or without an equation number. Appendix B defines special control sequences that will cover most of the remaining cases:

1. Two or more equations that should be aligned on `=` signs. (The alignment can also be on other signs like `≤`, etc.) For this case, type

$$\begin{aligned} & \text{\$}\text{\$}\text{\backslash eqalign}\{(\text{left-hand side}_1)\text{\textcircled{}}(\text{right-hand side}_1)\text{\backslash cr} \\ & \qquad \qquad \qquad (\text{left-hand side}_2)\text{\textcircled{}}(\text{right-hand side}_2)\text{\backslash cr} \\ & \qquad \qquad \qquad \vdots \\ & \qquad \qquad \qquad (\text{left-hand side}_n)\text{\textcircled{}}(\text{right-hand side}_n)\text{\backslash cr}\text{\$}\text{\$} \end{aligned}$$

with an optional `\eqno(formula)` just before the closing `"$$"` and after the closing `"\cr}"`. *N.B.: Don't forget to type the final `\cr`!* The relation symbols on which

you are aligning should be the first symbols of the right-hand sides (not the last symbols of the left-hand sides). If `\eqno` appears, the equation number will be centered vertically in the display (or—if it doesn't fit—it will be dropped down to the line following the display, as mentioned earlier). For example, if you type

```


$$\begin{aligned} a_1 + b_1 w + c_1 w^2 &= \alpha + \beta; \\ b_2 x + c_2 x^2 &= 0. \end{aligned} \eqno (30)$$


```

the result will be

$$\begin{aligned} a_1 + b_1 w + c_1 w^2 &= \alpha + \beta; \\ b_2 x + c_2 x^2 &= 0. \end{aligned} \tag{30}$$

Note that the left-hand sides are right-justified and the right-hand sides are left-justified, so the = signs line up; the whole formula is also centered, and the equation number (30) is halfway between the lines.

Ⓜ Sometimes you may want more or less vertical space between the aligned equations. Type `"\noalign{\vskip<glue>}"` after any `\cr`, to insert a given amount of extra glue after any particular equation line. (You can even do this before the first equation and after the last one.)

Ⓜ In general, the result of `\eqalign` is a `\vcentered` box, so `\eqalign` can be used in a fashion analogous to `\pile` or `\cpile`. Thus, it is possible to type such things as `"\eqalign{...} \quad \eqalign{...}"`, obtaining a display with two columns of aligned formulas.

2. Two or more equations that should be aligned, some of which have equation numbers. For this case you use `\eqalignno`, which is something like `\eqalign`, but each line now has the form

`<left-hand side>Ⓜ<right-hand side>Ⓜ<equation number>\cr .`

For example,

```


$$\begin{aligned} a_1 + b_1 w + c_1 w^2 &= \alpha + \beta; \tag{29} \\ b_2 x + c_2 x^2 &= 0. \tag{30} \end{aligned}$$


```

yields

$$\begin{aligned} a_1 + b_1 w + c_1 w^2 &= \alpha + \beta; & (29) \\ b_2 x + c_2 x^2 &= 0. & (30) \end{aligned}$$

You can't use `\eqno` together with `\eqalignno`; the equation numbers now must appear as shown.

If the `\eqno` of some line is blank, you can omit the `\eqno` before it. Example:

```



$$\begin{aligned} f(x) &= (x-1)(x+1) \\ &= x^2 - 1. \end{aligned} \tag{31}$$


```

This will produce the following display:

$$\begin{aligned} f(x) &= (x-1)(x+1) \\ &= x^2 - 1. \end{aligned} \tag{31}$$

(Note the position of the equation number.)

 You can use `\noalign` within `\eqalignno` to insert new lines of text. For example, `\noalign{\hjust{implies}}` will insert a line containing the word "implies" (at the left margin) between two aligned formulas.

3. A long equation that must be broken into two lines. You may want to type this as

```


$$\text{\twoline}{\text{first line}}{\text{glue}}{\text{second line}}$$


```

The formula's first line will be moved to the left so that it is one text size quad from the left margin, and its second line will be moved to the right so that it is one text size quad from the right margin. The specified glue will be inserted between these two lines in addition to the normal glue.

Another way to break a long equation is to use `\eqalign` with appropriate quads inserted at the beginning of the second line.

For example, here's an equation that is clearly too big to fit:

$$\sigma(2^{34} - 1, 2^{35}, 1) = -3 + (2^{34} - 1)/2^{35} + 2^{35}/(2^{34} - 1) + 7/2^{35}(2^{34} - 1) - \sigma(2^{35}, 2^{34} - 1, 1).$$

Let's break it just before the "+ 7". One way to do this is to type

```


$$\begin{aligned} \sigma(2^{34}-1, 2^{35}, 1) &= -3 + (2^{34}-1) / \\ &2^{35} + 2^{35} / (2^{34}-1) \tag{2pt} \{ \null 1 + 7 / 2^{35} \\ &\{ 35 \} (2^{34}-1) - \sigma(2^{35}, 2^{34}-1, 1) . \end{aligned}$$


```

The two-line result will then be

$$\sigma(2^{34} - 1, 2^{35}, 1) = -3 + (2^{34} - 1)/2^{35} + 2^{35}/(2^{34} - 1) + 7/2^{35}(2^{34} - 1) - \sigma(2^{35}, 2^{34} - 1, 1).$$

The other alternative is to type

```


$$\begin{aligned} \sigma(2^{34}-1, 2^{35}, 1) &= -3 + (2^{34}-1)/ \\ &2^{35} + 2^{35}/(2^{34}-1) \\ &+ 7/2^{35}(2^{34}-1) - \sigma(2^{35}, 2^{34}-1, 1). \end{aligned}$$


```

which yields

$$\sigma(2^{34} - 1, 2^{35}, 1) = -3 + (2^{34} - 1)/2^{35} + 2^{35}/(2^{34} - 1) + 7/2^{35}(2^{34} - 1) - \sigma(2^{35}, 2^{34} - 1, 1).$$

A couple of things should be explained and emphasized about this example: (a) The second line starts with “\null+7” instead of just “+7”. The control sequence \null is defined in Appendix B to mean a box of size zero, containing nothing, and this may seem rather insignificant; but it makes a big difference to T<sub>E</sub>X, because a plus sign in the middle of a formula is followed by a space, but a plus sign that begins a formula is not. Thus, you should always remember to type “\null” when you are continuing a multi-line formula. (b) When you use \twoline, never hit (carriage-return) on your keyboard just after the } that follows the (first line), or just after the } that follows the (glue); this (carriage-return) makes T<sub>E</sub>X think a space was intended, and \twoline won't work correctly. (You'll probably get some inscrutable error message like “\halign in display math mode must be followed by \$\$.”)

Breaking of long displayed formulas into several lines is an art; T<sub>E</sub>X never attempts to do it, because no set of rules is really adequate. The author of a mathematical manuscript should really decide how all such formulas should break, since the break position depends on subtle factors of mathematical exposition. Furthermore, different publishers tend to have different styles for line breaks. But several rules of thumb can be stated, since they seem to reflect the best mathematical practice:

- a) Although formulas within a paragraph always break *after* binary operations and relations, displayed formulas always break *before* binary operations and relations. Thus, we didn't end the first line of the above example with " $(2\uparrow\{34\}-1)+\backslash null$ ", we ended it with " $(2\uparrow\{34\}-1)$ " and began the second line with " $\backslash null+7$ ".
- b) The `\twoline` form is generally preferable for equations with a long left-hand side; then the break usually comes just before the  $=$  sign.
- c) When an equation is broken before a binary operation, the second line should start at least two quads to the right of where the innermost subformula containing that binary operation begins on the first line. For example, if you wish to break

$$\begin{aligned}
 & \sum_{1 \leq k \leq n} \left( \langle \text{formula}_1 \rangle + \langle \text{formula}_2 \rangle \right)
 \end{aligned}$$

at the plus sign between  $\langle \text{formula}_1 \rangle$  and  $\langle \text{formula}_2 \rangle$ , it is almost mandatory to have the plus sign on the second line appear somewhat to the right of the large left parenthesis corresponding to "`\left`". [Note further that your uses of `\left` and `\right` must balance in both parts of the broken formula. You could type, for instance,

$$\begin{aligned}
 & \sum_{1 \leq k \leq n} \left( \langle \text{formula}_1 \rangle \right. \\
 & \quad \left. + \langle \text{formula}_2 \rangle \right)
 \end{aligned}$$

provided that  $\langle \text{formula}_1 \rangle$  and  $\langle \text{formula}_2 \rangle$  are both of the same height and depth so that the `\left` on the first line will turn out to be the same size as the `\right` on the second. But in such cases it's simpler and safer to use, e.g., `\bigglp` and `\biggrp` instead of `\left` and `\right`.)

### <20> Definitions (also called macros)

You can often save time typing math formulas by defining control sequences as abbreviations for constructions that occur frequently in a particular manuscript. For example, if some manuscript frequently refers to the vector " $(x_1, \dots, x_n)$ ", you can type


$$\def\vec{x}\{(x_1, \dots, x_n)\}$$


and `\xvec` will henceforth be an abbreviation for "`(x↓1, \ldots, x↓n)`". Formulas like


$$\sum_{(x_1, \dots, x_n) \neq (0, \dots, 0)} (f(x_1, \dots, x_n) + g(x_1, \dots, x_n))$$

can then be typed simply as

```
$$\sum_{\xvec \neq (0, \ldots, 0)} \bigl( f \xvec + g \xvec \bigr)$$ .
```

 **TeX's** definition facility is what a designer uses to define all the standard formats, so Appendices B and E contain many illustrations of the use of `\def`. For example, `\eqalign` and `\eqalignno` are both defined in Appendix B. Defined control sequences can be followed by arguments, so we shall study the general rules for such definitions in this chapter. It's a good idea for you to look at Appendix B now.

 The general form is "`\def<controlseq><parameter text><result text>`", followed by an optional space, where the `<parameter text>` contains no `{` or `}`, and where all occurrences of `{` and `}` in the `<result text>` are properly nested in groups. Furthermore the `#` symbol (or whatever symbol is being used to stand for parameters, cf. Chapter 7) has a special significance: In the `<parameter text>`, the first appearance of `#` must be followed by 1, the next by 2, and so on; up to nine `#`'s are allowed. In the `<result text>` each `#` must be followed by a digit that appeared after `#` in the `<parameter text>`, or else the `#` should be followed by another `#`. The latter case stands for insertion of a single `#` in the result of any use of the definition; the former case stands for insertion of the corresponding argument.

 For example, let's consider a "random" definition that doesn't do anything useful except that it does exhibit **TeX's** rules. The definition

```
\def\cs AB#1#2CD\$\$#3 {#3{ab#1}#1 c\x ##2}
```

says that the control sequence `\cs` is to have a parameter text consisting of ten tokens

```
A, B, #1, #2, C, D, \$, E, #3, \,
```

and a result text consisting of twelve tokens

```
#3, {, a, b, #1, }, #1, \, c, \x, #, #2.
```

Henceforth when **TeX** reads the control sequence `\cs` it expects that the next two input tokens will be A and B (otherwise you will get the error message "Use of `\cs` doesn't match its definition"); then comes argument #1, then argument #2, then C, then D, then `\$`, then E, then argument #3, and finally a space. (It is customary to use the word "argument" to mean the string of tokens that gets substituted for a parameter; parameters appear in a definition, and arguments appear when that definition is used.)



Ⓔ How does T<sub>E</sub>X determine where an argument stops, you ask. Answer: If a parameter is followed in the definition by another token, the corresponding argument is the shortest (possibly empty) sequence of tokens with properly nested { . . . } groups that is followed in the input by this particular token. Otherwise the corresponding argument is the shortest *nonempty* sequence of tokens with properly nested { . . . } groups; namely, it is the next token, unless the token is {, when the argument is an entire group. In any case, if the argument found in this way has the form "{(balanced tokens)}", where (balanced tokens) stands for a sequence of tokens that is properly nested with respect to { and }, the outermost { and } enclosing this argument are removed. For example, let's continue with \cs as defined above, and suppose that the subsequent input contains

```
\cs AB{\Look}ABCD\$ E{And }{look} F.
```

Argument #1 will be the token \Look, since #1 is immediately followed by #2 in the definition, and since {\Look} is the shortest acceptable sequence of tokens following "\cs AB". Argument #2 will be the two tokens "AB", since it is to be followed by "C". Argument #3 will be the twelve tokens "{And }{look}", since it is to be followed by a space. Note that the exterior { and } are not removed from #3 as they were from #1, since that would leave an unnested string "And }{look". Note also that the space following "\$" is ignored since it isn't really a space (it follows a control sequence). The net effect then, after substituting arguments for parameters in the result text, will be that T<sub>E</sub>X's input will essentially become

```
{And }{look}{ab\Look}\Look□c\x#ABF.
```

The space □ here *will* be digested, even though it follows the control sequence \Look, because it was part of the defined result text. The "F." here comes from the yet-uns scanned input.

Ⓔ Definitions are not "expanded" (i.e., replaced by the result text) when they occur in a \def or an argument. Thus \Look and \\$ and \x are treated as single tokens in the example above, even though \Look has presumably been defined elsewhere. If @ or \cr occurs without being enclosed in { . . . }, in a definition or an argument in the midst of an alignment, T<sub>E</sub>X assumes that this @ or \cr belongs to the alignment and not to the definition or argument.

Ⓔ If you have difficulty understanding why some \def doesn't work as you expected, try running your program with \trace^355 (see Chapter 27).

Ⓔ The effect of \def lasts only until the control sequence is redefined or until the end of the group containing that \def. But there is another control sequence \gdef that makes a "global" definition, i.e., it defines a control sequence valid in all groups unless redefined. The \gdef instruction is especially useful in connection within \output routines, as explained in Chapter 23.

Exercise 20.1: The example definition of `\cs` includes a `##` in its result text, but the way `##` is actually used in that example is rather pointless. Give an example of a definition where `##` serves a useful purpose.

## <21> Making boxes

In Chapters 11 and 12 we discussed the idea of boxes and `glue`; now it is time to study the various facilities  $\TeX$  has for making various kinds of boxes. In most cases, you can get by with boxes that  $\TeX$  manufactures automatically with its paragraph builder, page builder, and math formula processor; but if you want to do nonstandard things, you have the option of making boxes by yourself.

To make a rule box, type `"\hrule"` in vertical mode or `"\vrule"` in horizontal mode, followed if desired by any or all of the specifications `"width(dimen)"`, `"height(dimen)"`, `"depth(dimen)"`, in any order. For example, you can type `"\vrule height 4pt width 3pt depth 2pt"` in the middle of a paragraph, and you will get the black box "█". The dimensions you specify should not be negative. If you leave any dimensions unspecified, you get the following by default:

	<code>\hrule</code>	<code>\vrule</code>
width	*	0.4 pt
height	0.4 pt	*
depth	0.0 pt	*

(Here "\*" means that the rule will extend to the boundary of the smallest enclosing box.)

To make a box from a horizontal list of boxes, type `"\hjust{<hlist>}"`, where `<hlist>` specifies the list of boxes in restricted horizontal mode. For example, `"\hjust{This is not a box}"` makes the box

This is not a box

(in spite of what it says). The boundary lines in this illustration aren't typeset, of course; they merely indicate the box's actual extent. The "just" in `\hjust` comes from the word "justification"—a printer's term, which is perhaps not completely justified in this context, but  $\TeX$  uses it anyway.

Similarly, the instruction “`\vjust{<vlist>}`” makes a box from a vertical list of boxes. If you type

```
\vjust{\hjust{T}\hjust{h}\hjust{i}\hjust{s}
       \hjust{ } \hjust{b}\hjust{o}\hjust{x}}
```

you will get this box:

Automatic baseline adjustment is done on vertical lists, as explained in Chapter 15. The following example shows what happens if the baseline adjustment is varied:

```
\vjust{\def\#1{\hjust{#1}}
       \baselineskip-1pt
       \lineskip 3pt
       \\T\\h\\i\\s
       \\{ }\\b\\o\\x}
```

Note that a specially defined control sequence `\\` saves a lot of typing in this example.

►Exercise 21.1: When the author of this manual first prepared the above example, he wrote “`\baselineskip 0pt`” instead of “`\baselineskip-1pt`”. Why didn’t this work?

►Exercise 21.2: How would you change the above example so that the letters are centered with respect to each other, instead of being placed flush left?

The phrase “list of boxes” in the above discussion really means a list of boxes and glue. But if `\hjust` and `\vjust` are used in the simple manner stated, the glue does not stretch or shrink. When you want the glue to do its thing, type “`\hjust to <dimen>{<hlist>}`” and you will get a box of the specified width; or type “`\vjust to <dimen>{<vlist>}`” and you will get a box of the specified height. (The depth of a `\vjusted` box is always the depth of the last box in the vertical list, except that it is zero when glue follows the last box.) You may also type “`\hjust to size{<hlist>}`” or “`\vjust to size{<vlist>}`”; this means that the `<dimen>` is to be the most recently specified `\hsize` or `\vsize`, respectively. Finally, there’s a further option of typing “`\hjust expand <dimen>{<hlist>}`” or “`\vjust expand <dimen>{<vlist>}`”; these expand the box to its natural width or height plus the nonnegative amount specified.

2 You can also get the effect of paragraphing and line-breaking with `\hsize`, in the following way: If you give the instruction "`\hjust to <dimen>`", and if at any time while processing the corresponding `<hlist>` the total natural width so far minus the total glue shrinkage so far exceeds the desired final width,  $\TeX$  will use its paragraph line-breaking routine to convert the horizontal list into one or more lines of the specified width. In this case the `\hjust` will actually result in a box formed from a vertical list of horizontal lists of the desired width.

For example, the box you are now reading was made by typing "`\hjust to 151pt{For example, the box ... five lines.}`" and  $\TeX$  broke it into five lines.

2 ▶Exercise 21.3: Look at the definition of `\spose` in Appendix B and explain why the glue "`\hskip Opt minus 100pt`" appears there. (This control sequence is for superposition; e.g., "`\spose xy`" superposes  $x$  on  $y$ . Examples of its use are given in Chapter 9 and Chapter 16.)

2 You can save a constructed box for later use by typing "`\save<digit><box>`", where `<digit>` is 0 or 1 or ... or 9 and `<box>` specifies a box. For example, "`\save3\hjust {The formula `` $x+y$ ``.}`" will save away the box

The formula " $x + y$ ".

(Note that math formulas are allowed in `<hlist>`s; but displays are not.) Later you can use this saved box by typing "`\box<digit>`". The `\save` and `\box` instructions are useful for constructing rather complex layouts like those in a newspaper page. Caution: You can use a saved box only once; after you type "`\box3`" the contents of box 3 becomes null. If you type "`\save3\box2`" the effect is to move box 2 to box 3 and then to make box 2 empty.

2 ▶Exercise 21.4: Define a control sequence `\boxit` so that "`\boxit<{<box>}>`" yields the given box surrounded by 3 points of space and ruled lines on all four sides. For example, this exercise has been typeset by telling  $\TeX$  to `\boxit{\boxit{\box4}}`, where box 4 was created by typing "`\save4\hjust to 300pt{\exno 21.4: Define ...}`".

Ⓡ To raise or lower a constructed box in a horizontal list, or in a math formula, precede it by “\raise(dimen)” or “\lower(dimen)”. For example, the  $\text{\TeX}$  control sequence that prints the  $\text{\TeX}$  logo in this manual is defined by

```
\def\TeX{\hjust{\:aT\hskip-2pt\lower1.94pt\hjust{E}\hskip-2pt X}}
```

Similarly, you can move a constructed box left or right in a vertical list if you type “\moveleft(dimen)” or “\moveright(dimen)” just before its description. The control sequences  $\text{\vcenter}$  and  $\text{\vtop}$  are also useful for box positioning (see Chapter 26).

Ⓡ There is also a way to repeat a box as many times as necessary to fill up some given space; this is what printers call “leaders.” The general construction is “\leaders(box or rule)(glue)”, where (box or rule) is any box or rule specified by  $\text{\hjust}$  or  $\text{\vjust}$  or  $\text{\box}$  or  $\text{\page}$  or  $\text{\hrule}$  or  $\text{\vrule}$ , and where (glue) is specified by  $\text{\hskip}$  or  $\text{\hfill}$  in horizontal mode,  $\text{\vskip}$  or  $\text{\vfill}$  in vertical mode.  $\text{\TeX}$  treats the glue in the normal way, possibly stretching it or shrinking it; but then instead of leaving the resulting space blank,  $\text{\TeX}$  places the contents of the box there, as many times as it will fit, subject to the condition that the reference point of each box will be congruent to some fixed number, modulo the box’s width (in horizontal leaders) or modulo the box’s height plus depth (in vertical leaders). This “congruence” means that leaders in different places will line up with each other. For example,

```
\def\lead{\leaders\hjust to 10pt{\hfill.\hfill}\hfill}
\hjust to size{Alpha\lead Omega}
\hjust to size{The Beginning\lead The Ending}
```

will produce the following two lines:

Alpha . . . . . Omega  
 The Beginning . . . . . The Ending

(Here “\hjust to 10pt{\hfill.\hfill}” specifies a box 10 points wide, with a period in its center; the control sequence  $\text{\lead}$  then causes this box to be replicated when filling another box.) When a rule is used as a leader, it completely fills the glue space; for example, if we had made the definition “\def\lead{\leaders\hrule\hfill}”, the two lines would have come out looking this way instead:

Alpha\_\_\_\_\_Omega  
 The Beginning\_\_\_\_\_The Ending

Leaders can be used in an interesting way to construct variable-width braces in the horizontal direction. T<sub>E</sub>X's math extension font `cmathx` (used with `basic` format) contains four characters that allow you to typeset such braces in the following way. First make the definitions

```
\def\bracex{\leaders\hrule height 1.5pt \hf111}
\def\dnbrace{${\char`772$\bracex${\char`775}
             {\char`774$\bracex${\char`773$}
\def\upbrace{${\char`774$\bracex${\char`773}
             {\char`772$\bracex${\char`775$}
```

Then

```
\hjust to 100pt{\dnbrace}
\hjust to 200pt{\upbrace}
```

will produce



This is occasionally useful in connection with math formulas.

► **Exercise 21.5:** How do you think the author of this manual made asterisks fill the rest of the current page? [Hint: The asterisk used (in font `cmr10`) has a height of 7.5 points.]



```
<< Alphabetical References >>
<< This package will produce an alphabetical list of LREFERS >>

page frame 60 high 77 wide
area text line 4 to 59
place text

.PREFACE 1;
.indent 0;
.single space
.nofill
.verbatim

.MACRO lrefer (name, argument, rprint, rcomment) $(
.send refx $(break; group)\name\argument$rprint$rcomment${apart}$
)$

.require "<aihpub>biblio.test" source!file;

.portion refx

.SINGLE SPACE
.TURN ON "{"

.AT "\" name "\" arg "$" rp "$" com "$" $(
.LREFER (name,
.largl,
.lrp1, com) {)$

.RECEIVE "\\\"
```

<< AIHANDBOOK BIBLIOGRAPHY >>  
<< NATURAL LANGUAGE SECTION >>

- LREFER (polya57, !Polya, #G. ##  
<<How to Solve It> (2nd ed.). New York: # Doubleday Anchor, 1957. !,  
!Polya, 1957!.)
- LREFER (minsky63, !Minsky, #M. ##  
Steps toward artificial intelligence. In E. #A. Feigenbaum & J. #Feldman #  
(Eds.), <<Computers and Thought>. New York: # McGraw-Hill, 1963. Pp. #406-450. !  
!Minsky, 1963!.)
- LREFER (FEIG63, !Feigenbaum, E. #A., & Feldman, J. (Eds.) ##  
<<Computers and Thought>. New York: # McGraw-Hill, 1963. !,  
!Feigenbaum & Feldman, 1963!.)
- LREFER (Har73, !Harris, L. R. ##  
The bandwidth heuristic search. <<IJCAI 3>, 1973, 23-29. !,  
!Harris, 1973!.)
- LREFER (Har74, !Harris, L. R. ##  
The heuristic search under conditions of error. ##  
<<Artificial Intelligence>, 1974, <<5>, 217-234. !,  
!Harris, 1974!.)
- LREFER (Poh170a, !Pohl, I. # <:2>##  
First results on the effect of error in heuristic search. In #  
B. Meltzer & D. Michie (Eds.), <<Machine Intelligence 5>. ##  
New York: American Elsevier, 1970, Pp. 219-236. (a) !,  
!Pohl, 1970a!.)
- LREFER (Poh170b, !Pohl, I. # <:3>##  
Heuristic search viewed as path finding in a graph. ##  
<<Artificial Intelligence>, 1970, <<1>, 193-204. (b) !,  
!Pohl, 1970b!.)
- LREFER (Poh173, !Pohl, I. # <:5>##  
The avoidance of (relative) catastrophe, heuristic competence,  
genuine dynamic weighting and computational issues in heuristic problem ##  
solving. <<IJCAI 3>, 1973, 12-17. !,  
!Pohl, 1973!.)
- LREFER (Mose67, !Moses, J. #  
<<Symbolic Integration>, MAC-TR-47, MAC Project, MIT, 1967. !,  
!Moses, 1967!.)
- LREFER (Slag61, !Slagle, #J. #R. <:1> ##  
<<A Heuristic Program that Solves Symbolic Integration Problems in #  
Freshman Calculus: # Symbolic Automatic Integrator (SAINT)>, 59-0001, #  
Lincoln Laboratory, MIT, 1961. !,  
!Slagle, 1961!.)
- LREFER (Slag63, !Slagle, #J. #R. <:2>##  
A heuristic program that solves symbolic integration problems in #  
freshman calculus. In E. #A. Feigenbaum & J. Feldman (Eds.), <<Computers #  
and Thought>. New York: # McGraw-Hill, 1963. Pp. 191-203. (Also in #  
<<JACM>, 1963, <<10>, 507-520. !),  
!Slagle, 1963!.)



.LREFER (newe76,  
.!Newell,#A.<:6>, & Simon,#H.#A. <+Computer science as empirical inquiry:#  
.Symbols and search>. The 1976 ACM Turing Lecture. <<CACM>, 1976, <<19>, 113-12  
.!Newell & Simon, 1976!)

18/1 break; group}\FEIG63\Feigenbaum, E.#A., & Feldman, J. (Eds.) ##<<Comput  
22/1 break; group}\Har73\Harris, L. R. ##The bandwidth heuristic search. <<IJCAI  
27/1 break; group}\Har74\Harris, L. R. ##The heuristic search under conditions o  
13/1 break; group}\minsky63\Minsky, #M. ##Steps toward artificial intelligence.  
49/1 break; group}\Mose67\Moses, J. ##<<Symbolic Integration>, MAC-TR-47, MAC Pro  
23/1 break; group}\newe76\Newell, #A. <:6>, & Simon, #H. #A. <+Computer science as  
33/1 break; group}\Poh170a\Pohl, I. # <:2>##First results on the effect of error  
38/1 break; group}\Poh170b\Pohl, I. # <:3>##Heuristic search viewed as path findi  
45/1 break; group}\Poh173\Pohl, I. # <:5>##The avoidance of (relative) catastroph  
8/1 break; group}\polya57\Polya, #G. ##<<How to Solve It> (2nd ed.). New York: # D  
55/1 break; group}\Slag61\Slagle, #J. #R. <:1> ##<<A Heuristic Program that Solves  
62/1 break; group}\Slag63\Slagle, #J. #R. <:2>##A heuristic program that solves sym

1970 END "COMPUTED!TEXT"a##{apart  
1970 END "COMPUTED!TEXT"b##{apart  
1963 END "COMPUTED!TEXT"##{apart

break; group)\FEIG63\Feigenbaum, E. #A., & Feldman, J. (Eds.) ##<<Computers  
break; group)\Har73\Harris, L. R. ##The bandwidth heuristic search. <<IJCAI  
break; group)\Har74\Harris, L. R. ##The heuristic search under conditions of  
break; group)\minsky63\Minsky, #M. ##Steps toward artificial intelligence. I  
break; group)\Mose67\Moses, J. #<<Symbolic Integration>, MAC-TR-47, MAC Proj  
break; group)\newe76\Newell, #A. <:6>, & Simon, #H. #A. <+Computer science as  
break; group)\Poh170a\Pohl, I. # <:2>##First results on the effect of error i  
break; group)\Poh170b\Pohl, I. # <:3>##Heuristic search viewed as path findin  
break; group)\Poh173\Pohl, I. # <:5>##The avoidance of (relative) catastrophe  
break; group)\polya57\Polya, #G. ##<<How to Solve It> (2nd ed.). New York: # D  
break; group)\Slag61\Slagle, #J. #R. <:1> ##<<A Heuristic Program that Solves S  
break; group)\Slag63\Slagle, #J. #R. <:2>##A heuristic program that solves symb

1 Mar 1979

8:54

ERRATA.TEX[TEX,DEK]

PAGE 2-1

This is a list of all errors in the September 1978 TEX user manual that were known on November 4, 1978. It also includes a few things that were omitted in September. All these changes (and only these) have been incorporated into the November 1978 manual.

Title page, change date to "November 1978 (second printing)" and change "draft" to "drafts" in the first line of the footnote.

Page 26, line 4, delete "of the second paragraph".

Page 29, line 13, change "later.)" to "later. A list of control sequences for special symbols appears in Appendix F.)

Page 40, line 15  
(one centimeter equals 26.600 Didot points)

Page 32, line 18, change "11" to "12".

Page 59, line 4, insert a ")" after this line.

Page 61, line 28, insert a "\$" before the pound sterling sign

Page 61, line 30, change ", and" to ", '\$\\\$', and"

Page 82, line 13, append this to the paragraph: "Another case is a formula like  $n/\log n$ , where a negative thin space has been inserted after the /.)".

Page 102, line 6, append this to the paragraph: "The control sequences `\vcenter` and `\vtop` are also useful for box positioning (see Chapter 26)."

Page 106, line 17, change "instead of '\$\ctr{#}\$'" to "instead of '\$\ctr{#\$}\$'".

Page 106, last three lines, change "processed; you might ... appropriate \def." to "processed."

Page 135, change the first 7 lines to the following:

based only on the current style, regardless of the sizes of numerator and denominator.

```

    / \vcenter \
  ◦ < { > { < \vlist > }          Append a centered or top-adjusted box.
    \ \vtop /
  
```

The specified vertical list is constructed in restricted vertical mode, then it is `\vjusted` and the resulting box is moved up or down so that (`\vcenter`) it is centered vertically just as large delimiters are, or (`\vtop`) the baseline of the topmost box in the vertical list coincides with the baseline of the formula. Then TEX resumes its activities in math mode.

Page 158, line 16, change ". (We" to ".\xskip (We".

Page 158, new paragraph inserted before the 4th-last line on this page:

- Within a paragraph, type '\$\xskip' before and after parenthesized sentences. (For example, there is an `\xskip` in the paragraph you are now reading, and in algstep E1 above.)

(The convention just explained has also been introduced into the entire TEX manual.)

Page 159, replace last two paragraphs by one paragraph, to wit:

If the exercise contains a 'hint' within a paragraph, you type '\$\xskip[{\s! Hint:} '\$; as usual, there should be no space before `\xskip`.

Page 162, line 4, change "Addison-Wesley's" to "the publisher's".

Page 165, line 15, change "4.5" to "4.625".

Page 165, line 23, change "4" to "4.25".

Page 182, line 6, change "or XYZages" to "XYZages, or XYZest"

Page 182, lines 14 and 15, change "-xe, or -xye, where x and y" to  
"-Xe, or -XYe, where X and Y"

Page 182, line 18, new sentence appended to this paragraph:  
"Similarly, final syllables of the form -Xed or -XYed (except -ized)  
are also disregarded."

Page 185, line 22, delete "guess-work"

Page 186, line 7, change "prob-a-bil-ity" to "prob-abil-ity".

Page 187, index entry for Bibliographic..., change "14" to "15".

Page 189, index entry for \deg, add page 164.

Page 195, delete 15th line in left-hand column.

Page 195, index entry for \spose, add page 39.

Page 196, index entry for \vcenter, add page 102.

Page 196, new index entry:  
\vtop (make <vlist> box using top baseline), 102, 135.

Page 196, index entry for \xskip, change "159" to "158--160".

Page 196, index entries for \yskip and \yyskip, change "158--159" and "158"  
to "159".

Page 197, index entry for \\$, add page 61.

Page 198, the TEX logo is too small, each time it appears on this page.

Extensions to TEX made since the November printing of the manual:

1. Several new <dimenparam>s have joined \hsize, \vsize, \topbaseline, etc., namely \lineskiplimit, \mathsurround, and \varunit.

By typing "\lineskiplimit <dimen>" you specify a dimension p such that \lineskip glue is used as the interline glue if and only if  $x-h-d < p$ , in the notation of Chapter 15.

By typing "\mathsurround <dimen>" you specify an amount of blank space to be inserted at the left and right of any formula embedded in text (i.e., formulas delimited by \$ and \$).

By typing "\varunit <dimen>" you specify the current value of a variable-size unit; the code "vu" denotes such relative units in a <dimen> specification. For example, after you define "\varunit 2pt", a <dimen> of "7vu" would stand for 14 points. When TEX begins, the values of \lineskiplimit, \mathsurround, and \varunit are 0pt, 0pt, and 1pt, respectively.

2. There is a new option to \advcount: If you type "\advcount <digit> by <number>" the specified counter is increased by the specified number. (When the "by" option is omitted, the counter is increased by plus-or-minus one as presently.) For example, "\advcount 0 by - \count 1" subtracts counter 1 from counter 0.

3. The control sequence \unskip can be used in horizontal mode (or restricted horizontal mode) to delete one glob of glue, if this glue was the last item added to the horizontal list. The main use of this is to remove an unwanted space that may have just appeared. For example, in a macro expansion the string "#1\unskip" denotes parameter #1 with a final blank space (or other glue) removed, if #1 ends with a blank space (or other glue).

4. Typing "\uppercase{<token list>}" in horizontal mode will change all lower-case letters of the token list into upper case. (But not the letters of control sequences.) Similarly, "\lowercase{<token list>}" changes upper-case letters into lower case.

5. Typing "\xdef<control sequence>{<result text>}" is like "\gdef<control sequence>{<result text>}" except that definitions in the result text are expanded. For example, "\xdef\z{\z\y}" will append the current result text of macro \y to the current result text of macro \z.

6. The new control sequence \ifpos is analogous to \ifeven; the \else code is evaluated only if the specified counter is zero or negative. For example, you can use \ifpos to test if a counter is zero in the following way:

```
\def\neg#1{\setcount#1-\count#1}
\def\ifzero#1#2\else#3{\ifpos#1{#3}\else{\neg#1
  \ifpos#1{\neg#1 #3}\else{\neg#1 #2}}
```

7. A new unit has been added: "em" equals one quad in the current font.

1 Mar 1979

8:54

ERRATA.TEX[TEX,DEK]

PAGE 4-1

Corrections noted since the November printing:

Page 14, line 15, change TEXes to TEXs.

Page 145, a new error message:

Warning: Long input line has been broken.

Your input file contained a very long sequence of characters between carriage returns. TEX arbitrarily broke it after 150 characters.

Page 166, in the definition of `\dimsectionbegin`:

Change `"\yyskip"` to `"\sectionskip"`.

Page 186, change `"com-put-a-*bil-ity"` to `"com-put-a*bil-ity"`.

Page 195, Spacing in math formulas,  
tables, 81, 83.



Important changes made to TEX on February 25, 1979:

The American Math Society will be printing copies of the TEX manual with all the above bugs cleaned up, and on this occasion it was the last chance to change TEX before changes became unwieldy. Thus, Knuth decided to make a couple improvements, to wit:

1. The control sequences `\hjust` and `\vjust` are henceforth changed to `\hbox` and `\vbox`. (This should cause you little or no trouble with MSs already typed, simply insert `"\def\hjust{\hbox}\def\vjust{\vbox}"` at the beginning of your file. The `basic.tex` file already has this, so if you are using basic format no change is necessary.)

2. The old kludge about `\hjust to ...{ }` making a boxed paragraph if the contents were too large has been replaced by a far better convention. This change will make TEX balk on some manuscripts it previously handled (e.g. it might now say "Overfull box, 1138.74 points too wide"), but only a few changes will really be necessary in your files.

Here are the new rules (replacing the previous rule on page 181):

\* You can also get the effect of paragraphing and line-breaking with `\hbox`, in the following way: If you give the instruction `"\hbox par<dimen>"`, TEX will use its paragraph line-breaking routine to convert the horizontal list into one or more lines of the specified width. In this case the `\hbox` will actually result in a box formed from a `{\sl vertical}` list of horizontal lists of the desired width. The boxed paragraph that you get is not indented.

For example, the box you are now reading was made by typing `"\hbox par 156pt{For example, the box ... five lines.}"` and TEX broke it into five lines.

\* If you specify hanging indentation with such a boxed paragraph, it applies to the box and not to the paragraph (if any) containing the box. For example,

`\hbox par 200pt{\hangindent 10 pt <text >}`

will put the specified text into a box 200 points wide, where all lines after the first are indented by 10 points at the left.

## <22> Alignment

A novice  $\TeX$  user can prepare manuscripts that involve mathematical formulas but no complicated tables; but a  $\TeX$  Master can prepare complicated tables using `\halign` or `\valign`. In this chapter, if you're ready for it, you can learn to be a  $\TeX$  Master. (And the next chapter—which talks about the design of `\output` routines—will enable you to become a Grandmaster.)

For simplicity, let's consider `\halign` first; `\valign` is similar, and it is used much more rarely. If you type

```
\halign to <dimen>{<alignment preamble>\cr<alignment entries>}
```

in vertical mode or restricted vertical mode, you append a list of aligned boxes that are each `<dimen>` units wide to the current vertical list; these boxes are formed from the `<alignment entries>` by using the specifications in the `<alignment preamble>`. We've already seen examples of alignment in Chapter 18, where `\halign` was used to construct matrices. In general, the preamble tells how to format individual vertical columns whose entries are going to be assembled into horizontal rows of the specified width. Before we get into any details of the alignment, let's observe straightaway that "`\halign to <dimen>`" can be changed to "`\halign to size`" if the `<dimen>` is to be the current `\hsiz`; it can be shortened to simply "`\halign`" if the minimum size (without shrinking) is desired, or replaced by "`\halign expand <dimen>`" if the boxes should be stretched to a given amount in addition to this minimum size. In other words, `\halign` has the same four options as `\hjust`.

The `<alignment preamble>` consists of one or more `<format>` specifications separated by `@`'s. Each `<format>` specification is a sequence of tokens that is properly nested with respect to `{ . . . }` groups and contains exactly one "#". For example, the `<alignment preamble>` suggested for three-column matrices in Chapter 18 was

```
 $\ctr{#}$\quad@ $\ctr{#}$\quad@ $\ctr{#}$
```

A `<format>` is essentially a simple `\def` with one parameter; the idea is to replace the # by whatever alignment entry is typed in that column position. For example, if the `<alignment entries>` following this preamble are

```
x+1@x+2@x+3\cr y+1@y+2@y+3\cr
```

then there will be two rows of the matrix obtained by substituting these entries in the preamble, namely

```
 $\ctr{x+1}$\quad    $\ctr{x+2}$\quad    $\ctr{x+3}$
 $\ctr{y+1}$\quad    $\ctr{y+2}$\quad    $\ctr{y+3}$
```

The  $\langle$ alignment entries $\rangle$  consist of zero or more  $\langle$ row $\rangle$ s; and a  $\langle$ row $\rangle$  is one or more entries separated by  $\otimes$ 's and followed by  $\backslash cr$ . In general if the preamble contains  $n$   $\langle$ format $\rangle$ s

$$\langle u_1 \rangle \# \langle v_1 \rangle \otimes \langle u_2 \rangle \# \langle v_2 \rangle \otimes \cdots \otimes \langle u_n \rangle \# \langle v_n \rangle$$


and if there are  $m$  rows each containing  $n$  entries


$$\begin{array}{ccccccc} \langle x_{11} \rangle & \otimes & \langle x_{12} \rangle & \otimes & \cdots & \otimes & \langle x_{1n} \rangle & \backslash cr \\ \langle x_{21} \rangle & \otimes & \langle x_{22} \rangle & \otimes & \cdots & \otimes & \langle x_{2n} \rangle & \backslash cr \\ \vdots & & \vdots & & & & \vdots & \\ \langle x_{m1} \rangle & \otimes & \langle x_{m2} \rangle & \otimes & \cdots & \otimes & \langle x_{mn} \rangle & \backslash cr \end{array}$$

we will obtain  $mn$  fleshed-out entries


$$\begin{array}{ccccccc} \langle u_1 \rangle \langle x_{11} \rangle \langle v_1 \rangle & \langle u_2 \rangle \langle x_{12} \rangle \langle v_2 \rangle & \cdots & \langle u_n \rangle \langle x_{1n} \rangle \langle v_n \rangle \\ \langle u_1 \rangle \langle x_{21} \rangle \langle v_1 \rangle & \langle u_2 \rangle \langle x_{22} \rangle \langle v_2 \rangle & \cdots & \langle u_n \rangle \langle x_{2n} \rangle \langle v_n \rangle \\ \vdots & \vdots & & \vdots \\ \langle u_1 \rangle \langle x_{m1} \rangle \langle v_1 \rangle & \langle u_2 \rangle \langle x_{m2} \rangle \langle v_2 \rangle & \cdots & \langle u_n \rangle \langle x_{mn} \rangle \langle v_n \rangle \end{array}$$


by repeatedly copying the preamble format information.



 Now here's what  $\text{\TeX}$  does with the  $mn$  fleshed-out entries: The natural width of each entry  $\backslash hjust\{\langle u_j \rangle \langle x_{ij} \rangle \langle v_j \rangle\}$  is determined; and the *maximum* natural width is computed in each column. Say  $w_j$  is the maximum natural width in the  $j$ th column; then each fleshed-out entry in that column is replaced by the box " $\backslash hjust\ to\ w_j\{\langle u_j \rangle \langle x_{ij} \rangle \langle v_j \rangle\}$ ". Thus, all entries in a particular column now have the same width. Finally these boxes are welded together to make the  $m$  rows, by inserting  $n+1$  elements of glue in each row (before the first box, between boxes, and after the last box). The glue to use in this welding process has previously been specified by " $\backslash tabskip\{glue\}$ ". The  $m$  row boxes are finally appended to the current vertical list.

 If you don't understand what was said so far, look back at the matrix example and reread the above until you understand. Because there are also some refinements that we shall now discuss. (a) After any  $\backslash cr$  you can type " $\backslash noalign\{\langle vlist \rangle\}$ ", and this  $\langle vlist \rangle$  will simply appear in its place among the aligned row boxes. The  $\langle vlist \rangle$  in this case usually contains vertical glue, penalty specifications, or horizontal rules; but it might contain anything that is allowed in restricted vertical mode, even another  $\backslash align$ . (b) If some row has fewer than  $n$  entries, i.e., if the  $\backslash cr$  of some row occurs before there have been  $n-1$   $\otimes$ 's, all remaining columns of the row are set to null boxes

regardless of their format. (This is not necessarily the same as “`\hjust{⟨u_j⟩⟨v_j⟩}`”; the preamble formats are simply ignored.) (c) If you specify `\tabskip⟨glue⟩` in the preamble, the  $n + 1$  globs of glue that weld together the final row boxes will be different, so you can get different spacing between columns. Here's how it works: The glue placed before column 1 is the `\tabskip` glue in effect when the `\halign` control sequence itself appears; the glue that replaces a `@` or `\cr` is the `\tabskip` glue in effect when that `@` or `\cr` appears in the preamble.



 Warning: Any spaces you type in the `⟨format⟩`s of the preamble will be taken seriously! Don't start a new line after a `@` unless you intend a corresponding space to be there in every column. (You may, of course, start a new line after `\cr` without inserting an unwanted space, or you can type “`@\!`” and go to a new line.) The same applies to spaces in the aligned entries; *always be extra careful with your use of spaces inside `\halign`.*

 Another warning: Don't use a construction like “`$$$`” in your `⟨format⟩`s if the corresponding column entries might be null. Otherwise `TEX` will scan “`$$$`” and think display math is intended, and this probably will lead to hopeless confusion. (The matrix example above has “`$_\ctr{#}$`” instead of “`\ctr{$$$}`” for precisely this reason. Another safe possibility would be “`\ctr{$_ $}`”.)

  You can have `\halign` or `\valign` within `\halign` or `\valign` (for example, matrices within aligned equations). In order to allow this, `TEX` insists that `{` and `}` be balanced in alignment entries, so that it is possible to distinguish which level of alignment corresponds to a given `@` or `\cr`. Consider, for example, the extremely simple alignment

$$\begin{array}{c} \backslash\halign{\backslashctr{#}\backslashcr \\ \langle\text{entry}\rangle\backslashcr} \end{array}$$

When `TEX` begins to scan the alignment entry, it scans the string of tokens “`\ctr{⟨entry⟩\cr} . . .`”; and the appearance of “`\ctr`” causes `TEX` to look for `\ctr`'s argument. This argument begins with “`{`”, so the scanning continues until the matching “`}`”. However, when the `\cr` is encountered after `⟨entry⟩`, `TEX` is supposed to insert the matching “`}`” from the preamble. If `⟨entry⟩` itself contains a use of `\halign`, there will be `\cr`'s in the middle of `⟨entry⟩`; so `TEX` doesn't simply look for the first `\cr`. Instead it ignores the tokens `@` and `\cr` until finding one that is not enclosed in braces, thereby correctly determining the argument to `\ctr`.

  Defined control sequences in the preamble are not usually expanded until the alignment entries are being processed. However, a control sequence following “`\tabskip⟨glue⟩`” in the preamble might be expanded, since a `⟨glue⟩` specification might involve control sequences. For example, “`\tabskip Opt \ctr{#}`” will effectively be

expanded by T<sub>E</sub>X to “\tabskip Opt \hfill # \hfill” while the preamble is being scanned, because T<sub>E</sub>X won't know (when it gets to “\ctr”) whether or not the expansion of this control sequence will begin with “plus 1pt” or some other continuation of the glue specification.

☞ In the rest of this chapter we shall discuss two worked-out examples. First suppose that we want to typeset three pairs of displayed formulas whose = signs are to be aligned, such as

$$\begin{aligned} V_i &= v_i - q_i v_j, & X_i &= x_i - q_i x_j, & U_i &= u_i, & \text{for } i \neq j; \\ V_j &= v_j, & X_j &= x_j, & U_j &= u_j + \sum_{i \neq j} q_i u_i. \end{aligned} \quad (13)$$

We could do this with three \equalign's, but let's not, since our current goal is to learn more about the general \halign construction. One solution is to type

```


$$\begin{aligned} &V_i = v_i - q_i v_j, & X_i &= x_i - q_i x_j, & U_i &= u_i, & \text{for } i \neq j; \\ &V_j = v_j, & X_j &= x_j, & U_j &= u_j + \sum_{i \neq j} q_i u_i. \end{aligned} \quad (13)$$


```

with some suitable alignment preamble. (It sometimes helps to figure out how you want to type the alignment entries before you design the preamble; there's a tradeoff between ease of typing the entries and ease of constructing the preamble.) One suitable preamble is

```

\rt{#}$@ \left{ $\null=#$ } \quad
@ $\rt{#}$@ \left{ $\null=#$ } \quad
@ $\rt{#}$@ \left{ $\null=#$ }

```

Note the \nulls here: they ensure proper spacing before the = signs, because the equations are being broken into two parts. Study this example carefully and you'll soon see how to make useful alignments.

☞ ▶Exercise 22.1: What would happen if “\vcenter” were replaced by “\vjust” in the above example?

☞ If we didn't have to include an equation number like “(13)”, the \vcenter could have been omitted; but then there would have been a possible page break between the two equations, and the equations would not have been centered on the line. (The only effect of the \$\$\$'s in “
$$\begin{aligned} &V_i = v_i - q_i v_j, & X_i &= x_i - q_i x_j, & U_i &= u_i, & \text{for } i \neq j; \\ &V_j = v_j, & X_j &= x_j, & U_j &= u_j + \sum_{i \neq j} q_i u_i. \end{aligned} \quad (13)$$
” is to insert \dispskip glue above and below the alignment.) One way to prevent a page break would be to insert “\noalign{\penalty 1000}” between the lines. And one way to center the equations would be to vary the tabskip glue, as in the definition of \equalignno in Appendix B. But it is much easier to use \vcenter.

Ⓔ The second example is slightly more complex, but once you master it you will have little or no trouble with other tables. Suppose you want to specify this:

AT&T Common Stock		
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
6	51-59	.95*

\*(first quarter only)

including all those horizontal and vertical lines. The table is to be 150 points wide. Here is one way to do it, letting the tabskip glue expand to give the column widths (so that, for example, the "Price" column will turn out to be exactly one en-dash-width wider than the "Year" column):

```

$$\vjust{\tabskip Opt
\def\|{\vrule height 9.25pt depth 3pt}
\def\.{\hskip-10pt plus 10000000000pt}
\hrule
\hjust to 150pt{\|\.AT&T Common Stock\.\|}
\hrule
\halign to 150pt{#\tabskip Opt plus 100pt
@ \hfill#@\#@\ctr{#}@\#@\hfill#@\#\tabskip Opt\cr
\|@\.Year\.\hfill@|\@\.Price\.\@|\@\.Dividend\.\hfill@|\|cr
\noalign{\hrule}
\|@1971@|\@41--54@|\@$2.60@|\|cr\noalign{\hrule}
\|@2@|\@41--54@|\@2.70@|\|cr\noalign{\hrule}
\|@3@|\@46--55@|\@2.87@|\|cr\noalign{\hrule}
\|@4@|\@40--53@|\@3.24@|\|cr\noalign{\hrule}
\|@5@|\@45--52@|\@3.40@|\|cr\noalign{\hrule}
\|@6@|\@51--59@|\@.95\spose*@\|cr\noalign{\hrule}}
\vskip 3pt
\hjust{*(first quarter only)}}$$

```

Ⓔ Here is an explanation of this rather long sequence of commands: The control sequence "\|" is defined to be a vertical rule that guarantees appropriate spacing of

baselines between individual rows. (TeX doesn't use `\baselineskip` and `\lineskip` before or after horizontal rules.) The alignment is defined in such a way that the `\tabskip` glue is zero at the left and right of the alignment, but it is "Opt plus 100pt" between columns; this glue will therefore expand to make the columns equally spaced. There are seven (not three) columns, since the vertical rules are considered to be columns. The preamble has just "#" for the columns that are to be vertical rules; the "Year" column and "Dividend" column both have format "`\hf111#`", causing them to be right-justified, while the "Price" column has format "`\ctr{#}`". The top row of the table appears before the `\halign`, since it does not have to be aligned with the other rows. In the second row of the table, an extra "`\hf111`" has been typed after "Year" and "Dividend", to compensate for the fact that the columns are being right-justified yet the titles are supposed to be centered. The special control sequence "`\.`" is also placed around these title words; this is somewhat tricky. It has the effect of telling TeX to ignore the width of the title words when computing the column widths. The asterisk in the final row of the table is preceded by "`\spose`" in order to make it zero-width, otherwise the decimal points wouldn't line up properly.

❖ Another way to get vertical and horizontal rules into tables is to typeset without them, then back up (using negative glue) and insert them.

❖ The control sequence `\valign` is analogous to `\halign`, but with rows and columns changing rôles. In this case `\cr` marks the bottom of a column. The boxes in each row will line up as if their reference points were at the bottom; in other words, their depth is effectively set to zero by modifying their height.

### <23> Output routines

We discussed TeX's page-building technique in Chapter 15. Constructed pages will be output directly, if the book design you are using has not specified any special `\output` routine. But usually a designer will have given special instructions that attach page numbers, headings, and so on. Even the basic format in Appendix B has a simple `\output` routine, described at the end of Chapter 15.

Complex `\output` specifications use the most arcane features of TeX, so it usually takes a designer three or four trials before he or she gets them right. Thus, you'll want to skip the rest of this chapter when you're first learning the TeX language. But—like alignments—`\output` routines soon lose their mystery after you have some experience with them.

When you type “`\output{<output list>}`”, the specified output list is stored away for later use, without expanding any of its defined control sequences. Then, when  $\TeX$  decides to output a page, the saved output list is effectively inserted into the input, wherever  $\TeX$  happens to be reading the input at the time. The purpose of the output list is to construct a box from a vertical list, as if one had typed

$$\backslash\text{vjust}\{\langle\text{output list}\rangle\};$$

this box is what gets output. The output routine might, however, produce a null box, if it saves away the current page in order to combine it with a later page.


This would be a good time for you to reread Chapter 15 if you don't recall  $\TeX$ 's mechanism for breaking pages. Since  $\TeX$  looks ahead for a good place to break—it usually is well into page 109, say, before page 108 is output—some care is needed to synchronize this asynchronous mechanism. For example, if you want to put the current section title at the top of each page, the section title might have changed by the time that page is actually shipped off to the output routine, since  $\TeX$  might be working on a new section before finding the most desirable break. An `\output` routine therefore needs some way of remembering past history. Such coordination is provided by so-called *marks*; when you're in vertical mode, you can type “`\mark{<mark text>}`”. This causes the `<mark text>` to be invisibly attached to your current position in the vertical list that is being broken into pages. If defined control sequences appear in a `<mark text>`, they are expanded at the time the mark appears, so that the `\output` routine will later be able to make use of values that were current.


The best way to think of this is probably to regard vertical mode as the mode in which you generate an arbitrarily long vertical list of boxes that somehow gets divided up into pages. The long vertical list may contain marks, and whenever you are outputting a page the `\output` routine will be able to make use of the most recent mark preceding the break at the bottom of the page (`\botmark`) and the most recent mark preceding the break at the top of the page (`\topmark`). For example, suppose your manuscript includes four instances of `\mark`, and suppose that the pages get broken in such a way that `\mark{a}` happens to fall on page 2, `\mark{\beta}` and `\mark{\gamma}` on page 4, and `\mark{\delta}` on page 5. Then

On page	<code>\topmark</code> is	<code>\botmark</code> is
1	null	null
2	null	<i>a</i>
3	<i>a</i>	<i>a</i>
4	<i>a</i>	$\gamma$
5	$\gamma$	$\delta$
6	$\delta$	$\delta$



The mark concept makes it possible to typeset things like dictionaries, where you want to indicate the current word-interval at the top of each page, if appropriate marks are inserted just before and after the space between entries.

 TeX has three control sequences that you are allowed to use only in `\output` routines: (i) `\page`, which represents the box containing the current page being output; (ii) `\topmark`, which represents the top mark for the current page (the corresponding mark text) is inserted into TeX's input at this point); (iii) `\botmark`, which is analogous to `\topmark`. The `\output` routine should use `\page` exactly once each time a page is to be output, but it may use `\topmark` and `\botmark` as often as desired.

 There are several other control sequences of special interest in connection with output routines, even though they are allowed to appear almost anywhere in a TeX manuscript:

`\setcount<digit><optional sign><number>` Sets one of ten "counters" to the specified number (possibly negative). For example, `"\setcount2 53"` sets counter number 2 equal to 53.

`\count<digit>` The current value of the specified "counter" is inserted into the input. If this number is zero, the result is the single digit "0"; if positive, the result is expressed as a decimal integer without leading zeros; if negative, the result is expressed as a roman numeral with lower case letters. (For example, `-18` yields `"xviii"`, `-19` yields `"xix"`.) As mentioned in Chapter 8, `\count<digit>` can also be used when TeX is expecting a `<number>`; for example, `"\setcount4\count2"` sets counter number 4 equal to the current contents of counter number 2.

`\advcount<digit>` The specified "counter" is increased by 1 if it is zero or positive, decreased by 1 if it is negative. (Thus, its magnitude increases by 1, but it retains the same sign.)

`\ifeven<digit><true text>\else<false text>` If the specified "counter" is even, the `<true text>` is input and the `<false text>` is ignored; if odd, the `<true text>` is ignored and the `<false text>` is input.

`\if<char12 If the input <char1 is equal to the input <char2, the <true text> is input and the <false text> is ignored; if not, the <true text> is ignored and the <false text> is input.`

Typical uses of `\if` have `<char1 constant, while <char2 is specified by a control sequence that has been defined elsewhere. For example, you might type`

```
\def \firsttime{T}
```

at the beginning of a chapter; then

```
\if T\firsttime{\gdef\firsttime{F}}\else{a}
```

will do *a* every time except the first, in each chapter. (Note that `\gdef` must be used here instead of `\def`, otherwise the new definition of `\firsttime` would be rescinded immediately!)

Now let's look at some examples. First, suppose you want your output pages to be numbered consecutively, with a number in font *c* centered at the bottom of each page. Suppose further that you want a running title in font *b* to be centered at the top of each page, except on the first page of each chapter. Each page (not counting margins) is to be  $4\frac{1}{2}$  inches wide and  $7\frac{1}{2}$  inches tall; but the pages output by TeX's page builder will have a height of  $6\frac{1}{2}$  inches and a maximum depth of  $\frac{1}{16}$  inch, so that you can put the running title in a half-inch strip at the top of each page, and you can put the current page number in a  $\frac{1}{16}$ - to  $\frac{1}{2}$ -inch strip at the bottom. Let's assume that font *z* is a big bold font suitable for chapter titles. Then the `\output` might be designed as follows:

```
\hsize4.5in\vsizer6.5in\maxdepth.0625in % inner page dimensions
\gdef\tpage{F} % \tpage will be T for title pages
\def\chapterbegin#1.#2{ % control sequence for new chapters
  \vfill\eject % finish previous chapter and begin a new page
  \gdef\tpage{T} % first page of chapter is a title page
  \vskip .5in % extra space above chapter title
  \ctrline{\:z Chapter #1.} % first line of title
  \vskip .25in % extra space between title lines
  \ctrline{\:z #2} % second line of title
  \vskip .5in % space between title and first paragraph
  \mark{#2} % insert a mark containing the running title
  \noindent % first paragraph will not be indented
  \tenpoint\!} % and it will use 10-point type fonts


\output{\vjust to 7.5in{ % begin output of 7.5-inch page
  \baselineskipOpt\lineskipOpt % turn off interline glue
  \if T\tpage{ % test if title page
    \gdef\tpage{F}\vskip.5in} % no running head on title page
  \else{\vjust to .15in{\vfill % fill space above running head
    \hjust to 4.5in{\hfill\:b\topmark\hfill}} % running head
    \vskip .35in} % space between running head and inner page
```


```


\page    % place the compiled inner page just below top strip
\vfill   % space between inner page and page number;
\hjust to 4.5in{\hfill\c\count0\hfill}}      % page number
\advcount0} % increase page number, end the \output routine

```

With this setup one types, for example, “\chapterbegin 13. {UNLUCKY NUMBERS}” at the beginning of chapter number 13. Appendix E shows how the more elaborate page layout of *The Art of Computer Programming* can be handled.

 ▶ **Exercise 23.1:** Why is it better for this \output routine to say “\hjust to 4.5in” than to say “\hjust to size”?

 ▶ **Exercise 23.2:** How would you change the above \output routine so that pages will come out with the top line of non-title pages saying “(page number)\_\_\_\_(running title)” on even-numbered pages and “(running title)\_\_\_\_(page number)” on odd-numbered pages? (Leave the page number at the bottom of title pages.)

 One more example should suffice to give the flavor of \output routines. Suppose you wish to typeset three-column format: three individual columns 6" tall by 1½" wide are to appear on a 7" × 5" page, with vertical rules between the columns. The page number is to be placed in the upper left corner of even-numbered pages and in the upper right corner of odd-numbered pages. For this application you should use \hsize 1.5in and \vsize 6in; and, say, \maxdepth .2in. (Recall that \maxdepth is the maximum amount by which the depth of the bottom line on a page is allowed to overhang the \vsize.) The \output routine has to save the first two “pages” it receives, then it must spew out three at once. There are at least two ways to do the job:

```

Solution 1. \output{\outa}
\def\outa{\output{\outb}\save1\page}
\def\outb{\output{\outc}\save2\page}
\def\outc{\output{\outa}
\vjjust to 7in{\baselineskipOpt\lineskipOpt
\vjust to 10pt{\vfill
\hjust to 5in{\:b
\ifeven0{\count0\hfill}\else{\hfill\count0}}}
\vfill
\hjust to 5in{\box1\hfill\vrule\hfill\box2
\hfill\vrule\hfill\page}}
\advcount0}

```

```

Solution 2. \def\firstcol{T}
             \output{\if T\firstcol{\gdef\firstcol{F}
             \gdef\secondcol{T}\save1\page}
             \else{\if T\secondcol{\gdef\secondcol{F}
             \save2\page}
             \else{\gdef\firstcol{T}
             \vjust to 7in{...(as before)...}\advcount0}}}}

```

Solution 1 is more elegant, but the switching mechanism of Solution 2 can be used in more complicated situations.

### <24> Summary of vertical mode

Now here is a complete specification of everything you are allowed to type in vertical mode. This chapter and the following two are intended to be a concise and precise summary of what we have been discussing rather informally. Perhaps it will be a useful reference when you're stuck and wondering what  $\TeX$  allows you to do.

Chapter 13 explains the general idea of vertical mode and restricted vertical mode. In both cases  $\TeX$  is scanning a “ $\langle vlist \rangle$ ” and building a vertical list containing boxes and glue; this list might also contain other things like penalty and mark specifications. The vertical list is empty when  $\TeX$  first enters vertical mode or restricted vertical mode, and it remains empty unless something is appended to it as explained in the rules below. For brevity the rules are stated for vertical mode; the same rules apply to restricted vertical mode unless the contrary is specifically stated.

When  $\TeX$  is in vertical mode, its next action depends on what it sees next, according to the following possibilities:

- $\langle space \rangle$  Do nothing.

This notation means: If  $\TeX$  is in vertical mode and you type a blank space, nothing happens and  $\TeX$  stays in vertical mode. (The end of a line in an input file counts as a blank space, and so do certain other characters, as explained in Chapter 7.)

- $\langle \backslash par \rangle$  Do nothing.

End of paragraph is ignored in vertical mode. This applies also to the “end of paragraph” signal that  $\TeX$  digests when you have blank lines in the input or at the end of a file page.

- $\langle \text{unknown control sequence} \rangle$  “! Undefined control sequence.”

For example, if you type “ $\backslash hjsut$ ” instead of “ $\backslash hjust$ ”, and if  $\backslash hjsut$  hasn't been defined, you get an error message showing that  $\backslash hjsut$  has just been scanned. To

recover you can type "i" (for insertion); then (when prompted by "\*") type "\hjust" and <carriage-return>, and TeX will resume as if the misspelling hadn't occurred.

- <defined control sequence> Macro call.

A control sequence that has been defined with \def or \gdef, for example a control sequence defined in a book format such as Appendix B or Appendix E, followed by its "arguments" (if any), will be replaced in the input as explained in Chapter 20.

- { Begin a new group.

A new level of nomenclature begins, as explained in Chapter 5; a matching } should appear later. The matching } usually occurs in vertical mode, but it might occur in horizontal mode (in the midst of some paragraph). The beginning of a new group does not affect the current vertical list.

- } End a group or an operation.

The matching { is identified, and all intervening \defs, \chcodes, \chpars, current font definitions, and glue parameter definitions are forgotten. If the matching { is the beginning of a group, TeX remains in vertical mode and the current vertical list is not affected. Otherwise TeX finishes whatever the { marked the beginning of, or you get an error message. The error messages are "Too many }'s", meaning that there was no matching {; or "Extra }", meaning that an unmatched right brace appears in the <v<sub>n</sub>> list of some alignment preamble; or "Missing \cr inserted", meaning that the matching { was in "\align(spec){". In the former cases the } is ignored; in the latter case a \cr is inserted.

- \hrule(rule spec) Append a horizontal rule.

The specified horizontal line is appended to the current vertical list. (See Chapter 21 for further details.) TeX remains in vertical mode.

- <box> Append a box.

Here <box> means one of the following:

\hjust(spec){<hlist>}	box formed in restricted horizontal mode
\vjust(spec){<vlist>}	box formed in restricted vertical mode
\box<digit>	saved box (e.g., \box1 was saved by \save1)
\page	current page (allowed only in output routines)

And <spec> is one of the following:

to <dimen>	desired width or height is specified
to size	width \hsize or height \vsize
<nothing>	use natural width or height
expand <dimen>	augment natural width or height

(Chapters 21 and 23 give further details.) The specified box is appended to the current vertical list, with appropriate interline glue depending on `\baselineskip` and `\lineskip` inserted just before it, as described in Chapter 15. (After using `\box` or `\page`, that `\box` or `\page` becomes null, so it can't be used twice.) Then  $\TeX$  resumes scanning in vertical mode.

- $\left\langle \begin{array}{l} \backslash\moveright \\ \backslashmoveleft \end{array} \right\rangle \langle \text{dimen} \rangle \langle \text{box} \rangle$  Append a shifted box.

The specified box is appended to the current vertical list as described above, but its contents are shifted left or right by the specified amount. (The right edge of the shifted box is used in figuring the maximum width of the box ultimately constructed from the current vertical list; but if the left edge of the appended box extends to the left of the current reference point, it will stick out of the constructed box.)

- `\save<digit><box>` Save a box.

The specified box is stored away for possible later use by "`\box<digit>`". Then  $\TeX$  resumes scanning in vertical mode, having made no change to its current vertical list.

- $\left\langle \begin{array}{l} \backslashvfill \\ \backslashvskip \end{array} \right\rangle \langle \text{glue} \rangle$  Append glue.

The specified glue is appended to the current vertical list. (See Chapter 12 for details about glue.)  $\TeX$  remains in vertical mode.

- `\leaders`  $\left\langle \begin{array}{l} \langle \text{box} \rangle \\ \langle \text{rule} \rangle \end{array} \right\rangle \left\langle \begin{array}{l} \backslashvfill \\ \backslashvskip \end{array} \right\rangle \langle \text{glue} \rangle$  Append leaders.

The specified leaders are appended to the current vertical list; this will have an effect like the specified glue except that the box or rule will be replicated in the resulting space (see Chapter 21).  $\TeX$  remains in vertical mode.

- `\noindent` Begin nonindented paragraph.

(Not allowed in restricted vertical mode.) The glue currently specified by `\parskip` is appended to the current vertical list. Then  $\TeX$  switches from page building to paragraph building by going into horizontal mode: What you type from now on until the next `\par` will be assembled into a paragraph and appended to the current vertical list.

- $\left\langle \begin{array}{l} \langle \text{char} \rangle \\ \langle \text{accent} \rangle \\ \$ \end{array} \right\rangle$  Begin indented paragraph.

(Not allowed in restricted vertical mode.) Here `<char>` stands for either `<letter>` or `<otherchar>` or `<nonmathletter>` or `\char<number>`, all of which are defined in Chapter 25. When any of these things occurs in vertical mode,  $\TeX$  thinks it is time to start a paragraph. The operations described above for `\noindent` are performed; then an

empty box whose width is the current value of `\parindent` is placed at the beginning of a horizontal list, which will become the next paragraph. Then processing continues as if the `<char>` or `<accent>` or `$` had appeared in horizontal mode. See Chapter 25 for a description of what happens next. (Note that a paragraph won't start with a box; if you really want to start a paragraph with a box, enclose it in `$'s`.)

- `\penalty<number>` Append a page break penalty.

(Has no effect in restricted vertical mode.) If the specified number is 1000 or more, page breaking is inhibited here; otherwise this number is added to the badness when deciding whether to break a page at this place. A negative penalty indicates a desirable place to break. (See Chapter 15.)  $\TeX$  remains in vertical mode.

- `\eject` Force a page break.

(Has no effect in restricted vertical mode.) A new page will start at this place in the current vertical list, no matter how "bad" it may be to break a page here. Two consecutive `\ejects` count as a single one.  $\TeX$  remains in vertical mode.

- `\mark<{<mark text>>` Append a mark.

(Not allowed in restricted vertical mode.) The mark text is attached invisibly to the current vertical list, with its defined control sequences expanded.  $\TeX$  remains in vertical mode.

- $\left\langle \begin{array}{l} \text{\topmark} \\ \text{\botmark} \end{array} \right\rangle$  Insert the text of a stored mark.

(Allowed only in `\output` routines.)  $\TeX$  inserts the specified mark text into its input; see Chapter 23.

- `\x` Extension to  $\TeX$ .

The control sequence `\x` allows special actions that might exist in some versions of  $\TeX$ . (Such extensions are obtained by loading a separately compiled module with the  $\TeX$  system; individual users might have their own special extension modules.)

- `\halign<spec>{<alignment preamble>\cr<alignment entries>>` Append alignment.

A vertical list of aligned rows is constructed as explained in Chapter 22, and this list is appended to the current list. Interline glue will be calculated as if the aligned boxes had been appended one by one in the ordinary way.

- $\left\langle \begin{array}{l} \otimes \\ \text{\cr} \end{array} \right\rangle$  Spurious alignment delimiter.

The symbols  $\otimes$  and `\cr` are detected deep inside  $\TeX$ 's scanning mechanism when they occur at the proper nesting level of braces, because they cause  $\TeX$  to start scanning a "`<vj>`" as explained in Chapter 22. Therefore if these symbols appear in vertical mode, they are ignored, and you get the error message "There's no `\halign` or `\valign` going on."

- `\ENDV` End of alignment entry.

An `\ENDV` instruction is inserted automatically by  $\TeX$  at the end of each “ $\langle v_j \rangle$ ” list of an alignment format. (You can't actually give this control sequence yourself; it only occurs implicitly.) If the alignment entry involves an unmatched `{`, you get the message “Missing } inserted.” Otherwise  $\TeX$  finishes processing this entry, by `\vjusting` the current vertical list, and appends the resulting box to the current column of the current `\valign`. (Interline glue is not used, but `\tabskip` glue will be inserted.) If the present `\ENDV` corresponds to an alignment entry that was followed by `\cr`,  $\TeX$  looks at the next part of the input as follows: Blank spaces are ignored; “`\noalign{<hlist>}`” causes the `<hlist>` to be processed in restricted horizontal mode, and the resulting horizontal list is appended to the horizontal list of the current `\valignment`; “`}`” terminates the `\valign`; and anything else is assumed to begin the next column of the alignment, so `\langle u_1 \rangle` is inserted into the input. On the other hand, if this `\ENDV` corresponds to an entry that was followed by `\@`,  $\TeX$  inserts `\langle u_{j+1} \rangle` into the input. In either case  $\TeX$  remains in restricted vertical mode to process the new alignment entry, beginning with an empty vertical list.

- $\left\langle \begin{array}{l} \text{\code{\topinsert}} \\ \text{\code{\botinsert}} \end{array} \right\rangle \langle \text{\code{vlist}} \rangle$  Floating insertion of a vertical list.

(Not allowed in restricted vertical mode.)  $\TeX$  reads the specified `\langle vlist \rangle` in restricted vertical mode and constructs the corresponding vertical list. This list will be inserted at the top or bottom of the next page on which it will fit, followed by `\topskip` glue or preceded by `\botskip` glue, respectively (see Chapter 15). If possible, two or more inserts will appear on the same page in first-in-first-out order. Note that stretchable or shrinkable glue in the vertical list is not set until the final page is made up. After the specified list has been constructed and stored in a safe place,  $\TeX$  resumes vertical mode where it left off.

- $\left\langle \begin{array}{l} \text{\code{\def}} \\ \text{\code{\gdef}} \end{array} \right\rangle \langle \text{\code{controlseq}} \rangle \langle \text{\code{parameter text}} \rangle \langle \text{\code{result text}} \rangle$  Define a control sequence.

The specified control sequence is defined as described in Chapter 20.  $\TeX$  remains in vertical mode, and the current vertical list is not affected. You are not allowed to redefine certain control sequences like `\:` and `\baselineskip`, because  $\TeX$  relies on these to control its operations at critical points. Definitions with `\def` disappear at the end of the current group; definitions with `\gdef` do not. It is best not to apply both `\def` and `\gdef` to the same control sequence in different parts of a manuscript.

- $\left\langle \begin{array}{l} \text{\code{\:}} \\ \text{\code{\mathex}} \end{array} \right\rangle \langle \text{\code{font}} \rangle$  Define the current font.

The specified font code is selected; “`\:`” selects the current font to be used in horizontal mode, as explained in Chapter 4, while “`\mathex`” selects the current `ex` font to be



used in mathematics mode, as explained in Chapter 18. If this code is making its first appearance in the manuscript it must be followed by the font file name (see Chapter 4 and Appendix S) followed by a space. Current font code selections are "local" and will be forgotten at the end of the current group.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- $\left\langle \begin{array}{l} \backslash\mathrm{mathrm} \\ \backslash\mathrm{mathit} \\ \backslash\mathrm{mathsy} \end{array} \right\rangle \langle \text{font} \rangle \langle \text{font} \rangle \langle \text{font} \rangle$  Define current math fonts.

The specified font codes are selected, providing up to three sizes of characters to be used in math formulas as explained in Chapter 18. If any font code is making its first appearance in the manuscript, it must be followed by the font file name (see Chapter 18 and Appendix S) followed by a space. Current font code selections are "local" and will be forgotten at the end of the current group.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- $\langle \text{dimenparam} \rangle \langle \text{dimen} \rangle$  Set a dimension parameter.

Here  $\langle \text{dimenparam} \rangle$  stands for one of the control sequences  $\backslash\text{hsize}$ ,  $\backslash\text{vsize}$ ,  $\backslash\text{maxdepth}$ ,  $\backslash\text{parindent}$ ,  $\backslash\text{topbaseline}$ . The corresponding  $\TeX$  parameter is set equal to the specified dimension;  $\TeX$  remains in vertical mode, and the current vertical list is not affected. This assignment is "global," it holds even after the end of a group. The initial default values of these five parameters are (324, 504, 3, 0, 10) points, respectively.

- $\langle \text{glueparam} \rangle \langle \text{glue} \rangle$  Define a glue parameter.

Here  $\langle \text{glueparam} \rangle$  stands for one of the control sequences  $\backslash\text{lineskip}$ ,  $\backslash\text{baselineskip}$ ,  $\backslash\text{parskip}$ ,  $\backslash\text{dispkip}$ ,  $\backslash\text{dispaskip}$ ,  $\backslash\text{dispbaskip}$ ,  $\backslash\text{topskip}$ ,  $\backslash\text{botskip}$ ,  $\backslash\text{tabskip}$ . The corresponding  $\TeX$  parameter is set equal to the specified glue;  $\TeX$  remains in vertical mode, and the current vertical list is not affected. This assignment is "local," it will be forgotten at the end of the current group. The initial value for all these types of glue is zero.

- $\backslash\text{chcode} \langle \text{number}_1 \rangle + \langle \text{number}_2 \rangle$  Define a character interpretation.

The character whose seven-bit code is  $\langle \text{number}_1 \rangle$  is subsequently treated as being of category  $\langle \text{number}_2 \rangle$ , where the category codes are described in Chapter 7. This definition will be local to the current group.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- $\backslash\text{chpar} \langle \text{number}_1 \rangle + \langle \text{number}_2 \rangle$  Define an integer parameter.

$\TeX$ 's internal parameter  $\langle \text{number}_1 \rangle$  is set equal to  $\langle \text{number}_2 \rangle$ . Here is a table of the

internal parameters:

Number	Name	Default value	Reference
0	<code>\trace</code>	345	Chapter 27
1	<code>\jpar</code>	2	Chapter 14
2	hyphenation	50	Chapter 14
3	doublehyphen	3000	Chapter 14
4	widowline	80	Chapter 15
5	brokenline	50	Chapter 15
6	binopbreak	95	Chapters 14&18
7	relbreak	50	Chapters 14&18
8	<code>\ragged</code>	0	Chapter 14
9	displaybreak	500	Chapter 15

This definition will be local to the current group.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- `\hangindent<dimen>`  $\left\langle \begin{array}{l} \text{for (number)} \\ \text{after (number)} \\ \text{(nothing)} \end{array} \right\rangle$  Set up hanging indentation.

This instruction causes a specified number of lines of the next paragraph to be indented either at the left margin or the right margin (see Chapter 14).  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- `\output{<vlist>}` Set the output routine.

The specified `<vlist>` is stored for later use when pages are output (see Chapter 23).  $\TeX$  remains in vertical mode, and the current vertical list is not affected. This assignment is "global," it will hold even after the end of the current group.

- `\setcount<digit>[optional sign]<number>` Set a specified counter.

One of ten counters, indicated by the specified digit, is set to the specified integer value (see Chapter 23). This assignment is "global," it is not rescinded at the end of a group.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- `\advcount<digit>` Advance the specified counter.

The magnitude of the specified counter is increased by 1.  $\TeX$  remains in vertical mode, and the current vertical list is not affected.

- `\count<digit>` Insert the specified counter.

The specified counter is converted to characters (see Chapter 23) and inserted into the input; this will cause  $\TeX$  to begin a new paragraph as explained earlier.

- $\left\langle \begin{array}{l} \text{\ifeven{digit}} \\ \text{\if{char}_1\{char}_2\} \end{array} \right\rangle \{ \langle \text{true text} \rangle \} \text{\else} \{ \langle \text{false text} \rangle \}$  Conditional text.

$\text{\TeX}$  reads either the true text or the false text, see Chapter 23.

- $\text{\input}$   $\langle \text{file name} \rangle \langle \text{space} \rangle$  Insert a file of text.

The specified file of characters is inserted into the input at this place. After the file has been read,  $\text{\TeX}$  will resume input at the present position (unless  $\text{\end}$  occurred in that file).

- $\text{\end}$  Stop.

(Not allowed in restricted vertical mode.) The current page is ejected, followed if necessary by pages containing leftover material, until there is nothing more to eject. Then if the last call on the output routine produced only a null box—for example, two out of three calls on the output routines at the end of Chapter 23 will do this—a page containing an empty box of size  $\text{\hsizex}\text{\vsize}$  is sent to the output routine, until either getting a nonnull output or until 25 consecutive null outputs have appeared. Then  $\text{\TeX}$  terminates: the output files are tidied up, and a friendly warning message is issued if there is an unmatched “{” still waiting for its “}”.

- $\text{\ddt}$  Print debugging data.

If bit 4 of the  $\text{\trace}$  parameter is 1,  $\text{\TeX}$  prints out its current activities (the lists and pages it is currently building). Furthermore if bit 40 of the  $\text{\trace}$  parameter is 1,  $\text{\TeX}$  will stop, giving you the chance to insert text on-line.  $\text{\TeX}$  remains in vertical mode, and the current vertical list is not affected.

- (anything else) “! You can't do that in vertical mode.”

If anything not listed above appears in vertical mode, you get an error message.  $\text{\TeX}$  ignores the token of input that broke the rules, and remains in vertical mode; the current vertical list is not affected.

## <25> Summary of horizontal mode

Here is a complete specification of everything you are allowed to type in horizontal mode. This chapter and the adjacent two are intended to be a concise and precise summary of what we have been discussing rather informally. Perhaps it will be a useful reference when you're stuck and wondering what  $\text{\TeX}$  allows you to do.

Chapter 13 explains the general idea of horizontal mode and restricted horizontal mode. In both cases  $\text{\TeX}$  is scanning an “ $\langle \text{hlist} \rangle$ ” and building a horizontal list containing boxes and glue; this list might also contain other things like penalty and insertion specifications. The horizontal list is empty when  $\text{\TeX}$  first enters horizontal mode or restricted horizontal mode, and it remains empty unless something is appended to it as

explained in the rules below. For brevity the rules are stated for horizontal mode; the same rules apply to restricted horizontal mode unless the contrary is specifically stated.

When  $\TeX$  is in horizontal mode, its next action depends on what it sees next, according to the following possibilities:

- $\langle$  (unknown control sequence)    “! Undefined control sequence.”

For example, if you type “ $r\A o l e$ ” instead of “ $r\A o l e$ ”, and if  $\A o l e$  hasn't been defined, you get an error message showing that  $\A o l e$  has just been scanned. To recover you can type “ $i$ ” (for insertion); then (when prompted by “ $*$ ”) type “ $\A o l e$ ” and  $\langle$ carriage-return $\rangle$ , and  $\TeX$  will resume as if the mistake hadn't occurred.

- $\langle$  (defined control sequence)    Macro call.

A control sequence that has been defined with  $\def$  or  $\gdef$ , for example a control sequence defined in a book format such as Appendix B or Appendix E, followed by its “arguments” (if any), will be replaced in the input as explained in Chapter 20.

- $\{$     Begin a new group.

A new level of nomenclature begins, as explained in Chapter 5; a matching  $\}$  should appear later. The matching  $\}$  usually occurs in horizontal mode, but it might occur in vertical mode (after the end of some paragraph). The beginning of a new group does not affect the current horizontal list.

- $\}$     End a group or an operation.

The matching  $\{$  is identified, and all intervening  $\def$ s,  $\chcodes$ ,  $\chpars$ , font definitions, and glue parameter definitions are forgotten. If the matching  $\{$  is the beginning of a group,  $\TeX$  remains in horizontal mode and the current horizontal list is not affected. Otherwise  $\TeX$  finishes whatever the  $\{$  marked the beginning of, or you get an error message. The error messages are “Too many  $\}$ ’s”, meaning that there was no matching  $\{$ ; or “Extra  $\}$ ”, meaning that an unmatched right brace appears in the  $\langle v_n \rangle$  list of some alignment preamble; or “Missing  $\backslash cr$  inserted”, meaning that the matching  $\{$  was in “ $\backslash align\langle spec \rangle \{$ ”. In the former cases the  $\}$  is ignored; in the latter case a  $\backslash cr$  is inserted.

- $\langle \begin{array}{l} \langle letter \rangle \\ \langle nonmathletter \rangle \\ \langle otherchar \rangle \end{array} \rangle$     Append a character box.

Here  $\langle letter \rangle$  normally means any of the characters  $A \dots Z$  and  $a \dots z$ , and  $\langle otherchar \rangle$  normally stands for any other character that has not been given a special meaning like the special meanings often assigned to  $\$$  and  $\&$  and  $\langle carriage-return \rangle$ , etc. However,  $\chcode$  can be used to reclassify any character, as explained in Chapter 7. A  $\langle nonmathletter \rangle$  is one of the control sequences  $\backslash ss$ ,  $\backslash ae$ ,  $\backslash AE$ ,  $\backslash oe$ ,  $\backslash OE$ ,  $\backslash o$ ,  $\backslash O$ , mentioned in Chapter 9. Each character has an associated 7-bit code that is used to select one of 128 characters from the current font. (If no current font has been defined, you lose:  $\TeX$  will come

to a grinding halt.) Information stored with the current font is now examined to see whether or not this character is the first of a ligature or kerned pair. If so,  $\text{\TeX}$  looks at the next character; when a ligature is completed, the two characters are replaced by a new character (as specified in the font) and this new character might in turn be the first of another ligature or kerned pair. In any event, a character box is appended to the current horizontal list; and if a kerned pair is found, appropriate negative glue is appended next, in such a way that the line-breaking and hyphenation algorithms will not be confused. Furthermore if the character code is '055 (the code for "-") or if a ligature ends with this particular code, a " $\backslash$ penalty 0" is automatically appended to the horizontal list.  $\text{\TeX}$  remains in horizontal mode.

- $\backslash$ char(number) Append a character box.

The (number) is reduced modulo 128, and  $\text{\TeX}$  proceeds just as if an (otherchar) had just been scanned having this 7-bit code.

- (accent)(accentee) Append an accented character.

Here (accent) stands for one of the control sequences  $\backslash$ ^,  $\backslash$ ~,  $\backslash$ A,  $\backslash$ v,  $\backslash$ u,  $\backslash$ =,  $\backslash$ "',  $\backslash$ H,  $\backslash$ b,  $\backslash$ s,  $\backslash$ t,  $\backslash$ a,  $\backslash$ l,  $\backslash$ c, discussed in Chapter 9, or for " $\backslash$ accent(number)"; and (accentee) stands for either (letter) or (nonmathletter) or (otherchar) or  $\backslash$ char(number), possibly preceded by a new font definition " $\backslash$ :{font}". The accent and accentee are made into character boxes, and the accent is superimposed on the accentee, moving the accent left or right if necessary so that it is centered (also taking into account the slantedness of the characters and their heights, based on information stored with the fonts). Furthermore the accent is raised or lowered in case the height of the accentee is different from the "xheight" of the accent's font (the height of lower case "x"). The width of the resulting box is the width of the accentee; this box is appended to the current horizontal list, and  $\text{\TeX}$  remains in horizontal mode.

- $\left\langle \begin{array}{l} \langle \text{space} \rangle \\ \backslash \square \end{array} \right\rangle$  Append variable space glue.

Here (space) means either an explicit typed space or an implicit one obtained at the end of a typed line. (Consecutive spaces are treated as single spaces, and spaces are sometimes ignored, as explained in Chapter 7.) The current font specifies what sort of glue should be inserted between words of a paragraph when they are typeset in that font. The stretchability and shrinkability of this glue is modified by the "space factor," as explained in Chapter 12, except that no modification is made when " $\backslash \square$ " has been typed.  $\text{\TeX}$  appends the glue to its current horizontal list and remains in horizontal mode.

- $\backslash$ quad Append one quad of space.

Space glue amounting to one quad in the current font is appended to the current horizontal list.  $\text{\TeX}$  remains in horizontal mode.

- `\!` Ignore space.

`TeX` looks at the next token of the input (expanding it if it is a defined control sequence), and discards it if it is a `<space>`. The current horizontal list is not affected, and `TeX` remains in horizontal mode.

- `\-` Append discretionary hyphen.

A “discretionary” hyphen is appended to the current horizontal list. This means that the current place is a legal place to break a line, with a specified penalty for hyphenation (see Chapter 14). If the line actually breaks here, character number ‘055 from the current font is inserted into the text, otherwise nothing is inserted. `TeX` remains in horizontal mode.

- `\/` Append italic correction.

If the final entry on the current horizontal list is not a character box, you get an error message

! Italic correction must follow an explicit character.

Otherwise an empty box whose width is the italic correction for the corresponding character is appended to the current horizontal list. (This information is stored in the font with each character, except in “`ex` fonts”; don’t try to use italic correction with a character from an `ex` font.) `TeX` remains in horizontal mode.

- `\vrule<rule spec>` Append a vertical rule.

The specified vertical line is appended to the current horizontal list. (See Chapter 21 for further details.) `TeX` remains in horizontal mode.

- `<box>` Append a box.

Here `<box>` means one of the following:

<code>\hjust&lt;spec&gt;&lt;{hlist}&gt;</code>	box formed in restricted horizontal mode
<code>\vjust&lt;spec&gt;&lt;{vlist}&gt;</code>	box formed in restricted vertical mode
<code>\box&lt;digit&gt;</code>	saved box (e.g., <code>\box1</code> was saved by <code>\save1</code> )
<code>\page</code>	current page (allowed only in output routines)

And `<spec>` is one of the following:

<code>to &lt;dimen&gt;</code>	desired width or height is specified
<code>to size</code>	width <code>\hsize</code> or height <code>\vsize</code>
<code>&lt;nothing&gt;</code>	use natural width or height
<code>expand &lt;dimen&gt;</code>	augment natural width or height

(Chapters 21 and 23 give further details.) The specified box is appended to the current horizontal list, and `TeX` resumes scanning in horizontal mode. (After using `\box` or `\page`, that `\box` or `\page` becomes null, so it can’t be used twice.)

- $\left\langle \begin{array}{l} \backslash\text{raise} \\ \backslash\text{lower} \end{array} \right\rangle \langle \text{dimen} \rangle \langle \text{box} \rangle$  Append a shifted box.

The specified box is appended to the current horizontal list as described above, but its contents are shifted up or down by the specified amount. (The top and bottom edges of the shifted box are used to compute the height and depth of the box ultimately constructed from the current horizontal list, as explained in Chapter 21.)

- $\backslash\text{save} \langle \text{digit} \rangle \langle \text{box} \rangle$  Save a box.

The specified box is stored away for possible later use by " $\backslash\text{box} \langle \text{digit} \rangle$ ". Then  $\text{\TeX}$  resumes scanning in horizontal mode, having made no change to its current horizontal list.

- $\left\langle \begin{array}{l} \backslash\text{hfill} \\ \backslash\text{hskip} \langle \text{glue} \rangle \end{array} \right\rangle$  Append glue.

The specified glue is appended to the current horizontal list. (See Chapter 12 for details about glue.)  $\text{\TeX}$  remains in horizontal mode.

- $\backslash\text{loaders} \left\langle \begin{array}{l} \langle \text{box} \rangle \\ \langle \text{rule} \rangle \end{array} \right\rangle \left\langle \begin{array}{l} \backslash\text{hfill} \\ \backslash\text{hskip} \langle \text{glue} \rangle \end{array} \right\rangle$  Append leaders.

The specified leaders are appended to the current horizontal list; this will have an effect like the specified glue except that the box or rule will be replicated in the resulting space (see Chapter 21).  $\text{\TeX}$  remains in horizontal mode.

- $\$(\text{formula})\$$  Append a math formula.

The specified  $\langle \text{formula} \rangle$  is scanned in math mode, as explained in Chapter 26. This results in a horizontal list, which is appended to the current horizontal list. Then  $\text{\TeX}$  resumes scanning in horizontal mode. Mathematics fonts (the so-called *rm* and *it* and *sy* and *ex* fonts) must have been defined earlier.

- $\backslash\text{par}$  End of paragraph.

(Ignored in restricted horizontal mode.) If the current horizontal list is empty, nothing happens. Otherwise the current horizontal list is "justified" using  $\text{\TeX}$ 's line-breaking routine described in Chapter 14; the resulting vertical list is appended to the current vertical list of the page-builder, and  $\text{\TeX}$  continues in vertical mode as described in Chapter 24.

- $\$\$(\text{display})\$\$$  Interrupt paragraph for display.

(Not allowed in restricted horizontal mode.) The current horizontal list is converted to a vertical list just as if a paragraph had ended, except that hanging indentation is not reset. Then the  $\langle \text{display} \rangle$  is processed, as explained in Chapter 26, resulting in another vertical list that is given to the page-builder. (A displayed formula counts as either two or three lines, with respect to the line count in hanging indentation, depending on whether  $\backslash\text{dispaskip}$  or  $\backslash\text{dispskip}$  glue is appended above the formula, cf. Chapter

19.) Then  $\TeX$  returns to horizontal mode, ignoring a space if it follows the closing "\$\$". At this point  $\TeX$ 's current horizontal list will be empty, so the paragraph will continue without indentation. Mathematics fonts (the so-called *rm* and *it* and *sy* and *ex* fonts) must have been defined earlier.

- `\penalty<number>` Append a line break penalty.

If the specified number is 1000 or more, line breaking is inhibited here; otherwise this number is added to the badness when deciding whether to break a line at this place. A negative penalty indicates a desirable place to break. (See Chapter 15.)  $\TeX$  remains in horizontal mode.

- `\eject` Force a page and line break.

(Forces only a line break when in restricted horizontal mode.) A new line will start at this place in the current horizontal list, and a new page will start with this new line when it is appended to the page builder's current vertical list, no matter how "bad" it may be to break a page or line here. (See the discussion in Chapter 14.)  $\TeX$  remains in horizontal mode.

- $\left\langle \begin{array}{l} \backslash\text{topmark} \\ \backslash\text{botmark} \end{array} \right\rangle$  Insert the text of a stored mark.

(Allowed only in `\output` routines.)  $\TeX$  inserts the specified mark text into its input; see Chapter 23.

- `\x` Extension to  $\TeX$ .

The control sequence `\x` allows special actions that might exist in some versions of  $\TeX$ . (Such extensions are obtained by loading a separately compiled module with the  $\TeX$  system; individual users might have their own special extension modules.)

- `\valign(spec){(alignment preamble)\cr(alignment entries)}` Append alignment. A horizontal list of aligned columns is constructed as explained in Chapter 22, and this list is appended to the current horizontal list. Then  $\TeX$  resumes scanning in horizontal mode.

- $\left\langle \begin{array}{l} \circ \\ \backslash\text{cr} \end{array} \right\rangle$  Spurious alignment delimiter.

The symbols  $\circ$  and `\cr` are detected deep inside  $\TeX$ 's scanning mechanism when they occur at the proper nesting level of braces, because they cause  $\TeX$  to start scanning a " $\langle v_j \rangle$ " as explained in Chapter 22. Therefore if these symbols appear in horizontal mode, they are ignored, and you get the error message "There's no `\halign` or `\valign` going on."

- `\ENDV` End of alignment entry.

An `\ENDV` instruction is inserted automatically by  $\TeX$  at the end of each " $\langle v_j \rangle$ " list of an alignment format. (You can't actually give this control sequence yourself; it only



occurs implicitly.) If the alignment entry involves an unmatched  $\{$ , you get the message "Missing  $\}$  inserted." Otherwise  $\TeX$  finishes processing this entry, by  $\hadjusting$  the current horizontal list, and appends the resulting box to the current row of the current  $\halign$ . (The  $\tabskip$  glue will also be inserted.) If the present  $\ENDV$  corresponds to an alignment entry that was followed by  $\cr$ ,  $\TeX$  looks at the next part of the input as follows: Blank spaces are ignored; " $\noalign\{<vlist>\}$ " causes the  $\langle vlist \rangle$  to be processed in restricted vertical mode, and the resulting vertical list is appended to the vertical list of the current  $\halignment$ ; " $\}$ " terminates the  $\halign$ ; and anything else is assumed to begin the next row of the alignment, so  $\langle u_1 \rangle$  is inserted into the input. On the other hand, if this  $\ENDV$  corresponds to an entry that was followed by  $\emptyset$ ,  $\TeX$  inserts  $\langle u_{j+1} \rangle$  into the input. In either case  $\TeX$  remains in restricted horizontal mode to process the new alignment entry, beginning with an empty horizontal list.

- $\left\langle \begin{array}{l} \backslash topinsert \\ \backslash botinsert \end{array} \right\rangle \langle \langle vlist \rangle \rangle$  Bound insertion of a vertical list.

(Not allowed in restricted horizontal mode.)  $\TeX$  reads the specified  $\langle vlist \rangle$  in restricted vertical mode and constructs the corresponding vertical list. This list will be inserted at the top or bottom of the same page on which the line containing the present place in the current horizontal list, followed by  $\topskip$  glue or preceded by  $\botskip$  glue, respectively. (See Chapter 15; this mechanism is intended primarily to accommodate illustrations and footnotes.) If necessary, two or more inserts will appear on the same page in first-in-first-out order. Note that stretchable or shrinkable glue in the vertical list is not set until the final page is made up. After the specified list has been constructed and stored in a safe place,  $\TeX$  resumes horizontal mode where it left off.

- $\left\langle \begin{array}{l} \backslash def \\ \backslash gdef \end{array} \right\rangle \langle \langle controlseq \rangle \langle parameter text \rangle \langle \langle result text \rangle \rangle \rangle$  Define a control sequence.

The specified control sequence is defined as described in Chapter 20. A  $\langle space \rangle$  following the definition will be ignored.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected. You are not allowed to redefine certain control sequences like  $\backslash :$  and  $\backslash baselineskip$ , because  $\TeX$  relies on these to control its operations at critical points. Definitions with  $\backslash def$  disappear at the end of the current group; definitions with  $\backslash gdef$  do not. It is best not to apply both  $\backslash def$  and  $\backslash gdef$  to the same control sequence in different parts of a manuscript.

- $\left\langle \begin{array}{l} \backslash : \\ \backslash mathex \end{array} \right\rangle \langle \langle font \rangle \rangle$  Define the current font.

The specified font code is selected; " $\backslash :$ " selects the current font to be used in horizontal mode, as explained in Chapter 4, while " $\backslash mathex$ " selects the current  $ex$  font to be used in mathematics mode, as explained in Chapter 18. If this code is making its first

appearance in the manuscript it must be followed by the font file name (see Chapter 4 and Appendix S) followed by a space. Current font code selections are "local" and will be forgotten at the end of the current group.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- $\left\langle \begin{array}{l} \backslash\mathrm{mathrm} \\ \backslash\mathrm{mathit} \\ \backslash\mathrm{mathbfy} \end{array} \right\rangle \langle \text{font} \rangle \langle \text{font} \rangle \langle \text{font} \rangle$  Define current math fonts.

The specified font codes are selected, providing up to three sizes of characters to be used in math formulas as explained in Chapter 18. If any font code is making its first appearance in the manuscript, it must be followed by the font file name (see Chapter 18 and Appendix S) followed by a space. Current font code selections are "local" and will be forgotten at the end of the current group.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- $\langle \text{dimenparam} \rangle \langle \text{dimen} \rangle$  Set a dimension parameter.

Here  $\langle \text{dimenparam} \rangle$  stands for one of the control sequences  $\backslash\text{hsize}$ ,  $\backslash\text{vsize}$ ,  $\backslash\text{maxdepth}$ ,  $\backslash\text{parindent}$ ,  $\backslash\text{topbaseline}$ . The corresponding  $\TeX$  parameter is set equal to the specified dimension;  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected. This assignment is "global," it holds even after the end of a group. The initial default values of these five parameters are (324, 504, 3, 0, 10) points, respectively.

- $\langle \text{glueparam} \rangle \langle \text{glue} \rangle$  Define a glue parameter.

Here  $\langle \text{glueparam} \rangle$  stands for one of the control sequences  $\backslash\text{lineskip}$ ,  $\backslash\text{baselineskip}$ ,  $\backslash\text{parskip}$ ,  $\backslash\text{dispkip}$ ,  $\backslash\text{dispaskip}$ ,  $\backslash\text{dispbskip}$ ,  $\backslash\text{topskip}$ ,  $\backslash\text{botskip}$ ,  $\backslash\text{tabskip}$ . The corresponding  $\TeX$  parameter is set equal to the specified glue;  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected. This assignment is "local," it will be forgotten at the end of the current group. The initial value for all these types of glue is zero.

- $\backslash\text{chcode} \langle \text{number}_1 \rangle + \langle \text{number}_2 \rangle$  Define a character interpretation.

The character whose seven-bit code is  $\langle \text{number}_1 \rangle$  is subsequently treated as being of category  $\langle \text{number}_2 \rangle$ , where the category codes are described in Chapter 7. This definition will be local to the current group.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- $\backslash\text{chpar} \langle \text{number}_1 \rangle + \langle \text{number}_2 \rangle$  Define an integer parameter.

$\TeX$ 's internal parameter  $\langle \text{number}_1 \rangle$  is set equal to  $\langle \text{number}_2 \rangle$ . Here is a table of the

internal parameters:

Number	Name	Default value	Reference
0	<code>\trace</code>	345	Chapter 27
1	<code>\jpar</code>	2	Chapter 14
2	hyphenation	50	Chapter 14
3	doublehyphen	3000	Chapter 14
4	widowline	80	Chapter 15
5	brokenline	50	Chapter 15
6	binopbreak	95	Chapters 14&18
7	relbreak	50	Chapters 14&18
8	<code>\ragged</code>	0	Chapter 14
9	displaybreak	500	Chapter 15

This definition will be local to the current group.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- `\hangindent(dimen)`  $\left\langle \begin{array}{l} \text{for (number)} \\ \text{after (number)} \\ \text{(nothing)} \end{array} \right\rangle$  Set up hanging indentation.

This instruction causes a specified number of lines of the next paragraph to be indented either at the left margin or the right margin (see Chapter 14).  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- `\output{<vlist>}` (optional space) Set the output routine.

The specified `<vlist>` is stored for later use when pages are output (see Chapter 23).  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected. This assignment is "global," it will hold even after the end of the current group.

- `\setcount<digit>` (optional sign) (number) Set a specified counter.

One of ten counters, indicated by the specified digit, is set to the specified integer value (see Chapter 23). This assignment is "global," it is not rescinded at the end of a group.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- `\advcount<digit>` Advance the specified counter.

The magnitude of the specified counter is increased by 1.  $\TeX$  remains in horizontal mode, and the current horizontal list is not affected.

- `\count<digit>` Insert the specified counter.

The specified counter is converted to characters (see Chapter 23) and inserted into the input;  $\TeX$  will read it in horizontal mode.

- $\left\langle \begin{array}{l} \text{\ifeven{digit}} \\ \text{\if{char}_1\{char}_2\} \end{array} \right\rangle \langle \text{\{true text\}} \rangle \text{\else} \langle \text{\{false text\}} \rangle$  Conditional text.

$\text{\TeX}$  reads either the true text or the false text, see Chapter 23. Spaces following the " $\langle \text{\{true text\}} \rangle$ " and " $\langle \text{\{false text\}} \rangle$ " are ignored.

- $\text{\addt}$  Print debugging data.

If bit 4 of the  $\text{\trace}$  parameter is 1,  $\text{\TeX}$  prints out its current activities (the lists and pages it is currently building). Furthermore if bit 40 of the  $\text{\trace}$  parameter is 1,  $\text{\TeX}$  will stop, giving you the chance to insert text on-line.  $\text{\TeX}$  remains in horizontal mode, and the current horizontal list is not affected.

- $\langle \text{\{anything else\}} \rangle$  "! You can't do that in horizontal mode."

If anything not listed above appears in horizontal mode, you get an error message.  $\text{\TeX}$  ignores the token of input that broke the rules, and remains in horizontal mode; the current horizontal list is not affected.

### <26> Summary of math mode

Here is a complete specification of everything you are allowed to type in math mode or display math mode. This chapter and the previous two are intended to be a concise and precise summary of what we have been discussing rather informally. Perhaps it will be a useful reference when you're stuck and wondering what  $\text{\TeX}$  allows you to do.

Chapter 13 explains the general idea of math mode and display math mode. In both cases  $\text{\TeX}$  is scanning an " $\langle \text{\mlist} \rangle$ " and building a horizontal list containing boxes, glue, and line-breaking information. The  $\langle \text{\mlist} \rangle$  is called a  $\langle \text{\display} \rangle$  if it is scanned in display math mode, a  $\langle \text{\formula} \rangle$  if scanned in ordinary math mode. Mathematics processing actually takes place in two stages: first the entire formula (up to the end of math mode) is input and made into a "tree structure," then this tree is converted into the desired horizontal list. The reason for doing the job in two steps is that  $\text{\TeX}$ 's language makes it impossible in general to determine the style for setting formulas as the formulas are being read in (e.g., a subsequent " $\text{\over}$ " might change everything). It is convenient, however, to describe the rules below as if  $\text{\TeX}$  had clairvoyance, knowing what style to use as it reads the input. Please keep in mind that the correct style will be chosen for subformulas, according to the rules in Chapters 17 and 18, even though the following description makes that seem somewhat miraculous. For brevity the rules below are stated for math mode; the same rules apply to display math mode unless the contrary is specifically stated.

When  $\text{\TeX}$  is in math mode, its next action depends on what it sees next, according to the following possibilities:

- `<space>` Do nothing.

This notation means: If  $\TeX$  is in math mode and you type a blank space, nothing happens and  $\TeX$  stays in math mode. (The end of a line in an input file counts as a blank space, and so do certain other characters, as explained in Chapter 7.)

- `<unknown control sequence>` “! Undefined control sequence.”

For example, if you type “`\alfa`” instead of “`\alpha`”, and if `\alfa` hasn't been defined, you get an error message showing that `\alfa` has just been scanned. To recover you can type “i” (for insertion); then (when prompted by “\*”) type “`\alpha`” and `<carriage-return>`, and  $\TeX$  will resume as if the mistake hadn't occurred.

- `<defined control sequence>` Macro call.

A control sequence that has been defined with `\def` or `\gdef`, for example a control sequence defined in a book format such as Appendix B or Appendix E, followed by its “arguments” (if any), will be replaced in the input as explained in Chapter 20.

- `<(mlist)>` Append a subformula.

The `<(mlist)>` is processed in math mode and `\hjusted` into a box having its natural width. This box is then appended to the current list as an “Ord” box. Definitions inside the subformula are forgotten afterwards.

- `\left<delim>(mlist)\right<delim>` Append a subformula with variable delimiters. The `<(mlist)>` is processed in math mode, and surrounded by delimiters of sufficient size to contain it, as explained in Chapter 18. The resulting list is `\hjusted` and appended to the current list as an “Ord” box. Definitions inside the subformula are forgotten afterwards.

- `>` “Extra `>`.”

The matching `<`, if any, lies outside the `$` or `\left` that precedes the current `<(mlist)>`, so an error message is issued and the `>` is ignored.

- `\right` “Extra `\right`.” or “Missing `>` inserted.”

The matching `\left`, if any, lies outside the `$` or `<` that preceded the current `<(mlist)>`, so an error message is issued.  $\TeX$  automatically inserts a “`>`” if it appears to be missing.

- `$` “Missing `\right` inserted.” or “Missing `>` inserted.”

The matching `$`, if any, lies outside the `\left` or `<` that preceded the current `<(mlist)>`, so an error message is issued and  $\TeX$  automatically inserts what it assumes was missing.

- $\left\langle \begin{array}{l} \langle \text{letter} \rangle \\ \langle \text{mathchar} \rangle \\ \langle \text{otherchar} \rangle \end{array} \right\rangle$  Append a character box.

Here `<letter>` normally means any of the characters `A . . . Z` and `a . . . z`, and `<otherchar>` normally stands for any other character that has not been given a special meaning like the special meanings often assigned to `$` and `@` and `<carriage-return>`, etc. However, `\chcode`

can be used to reclassify any character, as explained in Chapter 7. A  $\langle\text{mathchar}\rangle$  is one of the many control sequences  $\backslash\alpha$ ,  $\backslash\beta$ , etc. listed in Appendix F. Each  $\langle\text{mathchar}\rangle$  has an associated 9-bit code that is used to select one of 512 characters from  $\text{\TeX}$ 's current math fonts in the desired size; each  $\langle\text{letter}\rangle$  and  $\langle\text{otherchar}\rangle$  also has an associated 9-bit code, determined from its 7-bit code by using a table in Appendix F. Each character also has an associated category (Ord or Op or Bin, etc.), as explained in Chapter 18 and Appendix F; these categories are used to determine spacing and line-breaking. The character box is appended to the current list and  $\text{\TeX}$  continues scanning in math mode. (Note: The italic correction is included when computing the width of this box. However, it will be removed by  $\text{\TeX}$  if this box has a subscript but no superscript; thus, subscripts will be closer to letters like "P". The spacing on  $\text{\TeX}$ 's math fonts is intended to make formulas look right when typeset by  $\text{\TeX}$ 's rules, so it is quite different from spacing that makes text look right; cf. the examples of fonts `cmil0` and `cmti10` in Chapter 18.)

- $\backslash\text{char}\langle\text{number}\rangle$  Append a character box.

The  $\langle\text{number}\rangle$  is reduced modulo 512, and  $\text{\TeX}$  proceeds just as if a  $\langle\text{mathchar}\rangle$  of category Ord has just been scanned having this 9-bit code.

- $\uparrow\langle\text{atom}\rangle$  Superscript the previous box.

(Here and in two rules that follow, an  $\langle\text{atom}\rangle$  is either a single character (i.e.,  $\langle\text{letter}\rangle$  or  $\langle\text{mathchar}\rangle$  or  $\langle\text{otherchar}\rangle$  or  $\backslash\text{char}\langle\text{number}\rangle$ ) or a subformula of the form " $\langle\text{mlist}\rangle$ ". Atoms may be regarded as rigid boxes that will be combined to build up larger formulas.) If the last element of the current list is not a box, append a null box. Otherwise if the last box of the current list has already been superscripted, report a "Double superscript" error. Attach the box corresponding to the  $\langle\text{atom}\rangle$  as the superscript of the last box of the current list.

- $\downarrow\langle\text{atom}\rangle$  Subscript the previous box.

Subscripting is entirely analogous to superscripting.

- $\left\langle\begin{array}{l} \langle\text{mathcontrol}\rangle \\ \langle\text{accent}\rangle \end{array}\right\rangle\langle\text{atom}\rangle$  Build up a formula.

Here  $\langle\text{mathcontrol}\rangle$  stands for one of the nine control sequences  $\backslash\text{sqrt}$ ,  $\backslash\text{underline}$ ,  $\backslash\text{overline}$ ,  $\backslash\text{mathop}$ ,  $\backslash\text{mathbin}$ ,  $\backslash\text{mathrel}$ ,  $\backslash\text{mathopen}$ ,  $\backslash\text{mathclose}$ ,  $\backslash\text{mathpunct}$ ; and  $\langle\text{accent}\rangle$  stands for one of the control sequences  $\backslash\grave{}$ ,  $\backslash\acute{}$ ,  $\backslash\grave{A}$ ,  $\backslash\grave{v}$ ,  $\backslash\grave{u}$ ,  $\backslash\grave{=}$ ,  $\backslash\grave{}$ ,  $\backslash\grave{H}$ ,  $\backslash\grave{b}$ ,  $\backslash\grave{s}$ ,  $\backslash\grave{t}$ ,  $\backslash\grave{a}$ ,  $\backslash\grave{l}$ ,  $\backslash\grave{c}$ , discussed in Chapter 9, or for " $\backslash\text{accent}\langle\text{number}\rangle$ ". (The  $\langle\text{number}\rangle$  in the latter case is reduced modulo 512.) Each of these does something to the box formed from the  $\langle\text{atom}\rangle$ :  $\backslash\text{sqrt}$  inserts a variable-size radical sign in front of the box and a line over the box (and a little blank space above that line);  $\backslash\text{underline}$  and  $\backslash\text{overline}$  insert a line and a little blank space under or over the box; the control sequences  $\backslash\text{mathop}$ , ...,  $\backslash\text{mathpunct}$  are simply used to classify the box as type Op,

..., Punct, respectively; and an accent is centered over the box. (Accents in horizontal mode are corrected for slant, but in math mode they are simply centered; in both cases they are raised or lowered by the same amount when applied to the same letter.) The box resulting from the specified operation is appended to the current list, and  $\TeX$  continues in math mode.

- $\langle \text{mathglue} \rangle$  Append glue based on the current style.

Here  $\langle \text{mathglue} \rangle$  means one of the control sequences  $\backslash,$ ,  $\backslash\sqcup$ ,  $\backslash\>$ ,  $\backslash;$ ,  $\backslash\text{quad}$ ,  $\backslash\geq$ ,  $\backslash!$ ,  $\backslash?$ ,  $\backslash<$ ,  $\backslash\leq$ , described in Chapter 18. The corresponding glue is appended to the current list, and  $\TeX$  continues in math mode.

- $\langle \begin{array}{l} \backslash\text{fill} \\ \backslash\text{skip}(\text{glue}) \end{array} \rangle$  Append explicit glue.

The specified glue is appended to the current list. (See Chapter 12 for details about glue, and see Chapter 17 for an example of  $\backslash\text{fill}$  used in the numerator of a formula.)  $\TeX$  remains in math mode.

- $\langle \text{box} \rangle$  Append a box.

Here  $\langle \text{box} \rangle$  means one of the following:

$\backslash\text{hjust}(\text{spec})\langle(\text{hlist})\rangle$	box formed in restricted horizontal mode
$\backslash\text{vjust}(\text{spec})\langle(\text{vlist})\rangle$	box formed in restricted vertical mode
$\backslash\text{box}(\text{digit})$	saved box (e.g., $\backslash\text{box}1$ was saved by $\backslash\text{save}1$ )
$\backslash\text{page}$	current page (allowed only in output routines)

And  $\langle \text{spec} \rangle$  is one of the following:

to $\langle \text{dimen} \rangle$	desired width or height is specified
to size	width $\backslash\text{hsize}$ or height $\backslash\text{vsize}$
$\langle \text{nothing} \rangle$	use natural width or height
expand $\langle \text{dimen} \rangle$	augment natural width or height

(Chapters 21 and 23 give further details.) The specified box is appended to the current list as an Ord box, and  $\TeX$  resumes scanning in math mode. (After using  $\backslash\text{box}$  or  $\backslash\text{page}$ , that  $\backslash\text{box}$  or  $\backslash\text{page}$  becomes null, so it can't be used twice.)

- $\langle \begin{array}{l} \backslash\text{raise} \\ \backslash\text{lower} \end{array} \rangle(\text{dimen})\langle \text{box} \rangle$  Append a shifted box.

The specified box is appended to the current list as described above, but its contents are shifted up or down by the specified amount.

- $\backslash\text{save}(\text{digit})\langle \text{box} \rangle$  Save a box.

The specified box is stored away for possible later use by " $\backslash\text{box}(\text{digit})$ ". Then  $\TeX$  resumes scanning in math mode, having made no change to its current list.

- `\*` Append discretionary times sign.

A “discretionary”  $\times$  is appended to the current list. This means that the current place is a legal place to break a line, with a specified penalty for hyphenation (see Chapter 14). If the line actually breaks here, character number ‘402 from the current font is inserted into the text; otherwise nothing is inserted.  $\TeX$  remains in math mode.

- `\limitswitch` Change convention on displayed limits.

(Allowed only when the last item in the current list is an Op box; has an effect only when setting a formula in display style.)  $\TeX$ 's normal convention for typesetting the “limits” (i.e., the superscript and subscript) of an operator in display style is to center them above and below the Op box—unless that Op box is a single character in the current `ex` font having a nonzero “italic correction” in the font; in the latter case the subscripts and superscripts are normally set to the right as usual. But `\limitswitch` has the effect of reversing these conventions on the current operator: centering changes to placement at the right and vice versa.  $\TeX$  remains in math mode.

- $\left\langle \begin{array}{c} \backslashover \\ \backslashabove(dimen) \\ \backslashatop \end{array} \right\rangle$  Separate numerator from denominator.

If a numerator has previously been set aside for the current formula, give an error message

! Ambiguous; you need another { and }.

and ignore the input. Otherwise the current list is set aside to be the numerator, and the list after this point until the end of the formula will be the denominator. Afterwards the numerator will be centered over the denominator, essentially by inserting the glue “`\hskip Opt plus 100000pt`” at the left and right of whichever one has less natural width and `\hjusting` it to the width of the other. The fraction line inserted between them will be at the height of the “axis” of the overall formula (a position specified in the `sy` font of the appropriate size). The current `ex` font specifies a “default rule thickness” to be used for the ruled lines in `\sqrt`, `\underline`, and `\overline`; this same thickness is used for the fraction line in `\over`, while `\above` lets you specify any desired thickness. (See the examples in Chapter 17.) The thickness is zero for `\atop`, i.e., there is no fraction line at all; in this case, the positioning of numerator and denominator is somewhat different in order to take advantage of the extra flexibility. A little extra space is attached to the left and right of the formula after the numerator and denominator have been pasted together.

- `\comb(delim)(delim)` Build a combinatorial formula.

This is like `\atop`, except that the specified delimiters are placed at the left and right of the formula after the numerator and denominator have been positioned. (In fact, “`\atop`” is precisely equivalent to “`\comb. .`”.)  $\TeX$  chooses the size of the delimiters based only on the current style, regardless of the sizes of numerator and denominator.



- $\langle \begin{array}{l} \backslash\text{center} \\ \backslash\text{vtop} \end{array} \rangle$  Append a centered or top-adjusted box.

The specified vertical list is constructed in restricted vertical mode, then it is  $\backslash\text{vjusted}$  and the resulting box is moved up or down so that ( $\backslash\text{center}$ ) it is centered vertically just as large delimiters are, or ( $\backslash\text{vtop}$ ) the baseline of the topmost box in the vertical list coincides with the baseline of the formula. Then  $\text{T}_{\text{E}}\text{X}$  resumes its activities in math mode.

- $\backslash\text{penalty}(\text{number})$  Append a line break penalty.  
(This has no effect in a subformula or a displayed formula.) If the specified number is 1000 or more, line breaking is inhibited here; otherwise this number is added to the badness when deciding whether to break a line at this place. A negative penalty indicates a desirable place to break. (See Chapter 15.) If this penalty is specified immediately following a Bin or Rel box, it overrides the penalty ordinarily placed there (see Chapter 18).  $\text{T}_{\text{E}}\text{X}$  remains in math mode.

- $\backslash\text{eject}$  Force a page and line break.  
(This has no effect in a subformula or a displayed formula.) A new line will start at this place in the current horizontal list, and a new page will start with this new line when it is appended to the page builder's current vertical list, no matter how "bad" it may be to break a page or line here. (See the discussion in Chapter 14.)  $\text{T}_{\text{E}}\text{X}$  remains in math mode.

- $\langle \text{mathstyle} \rangle$  Define the current style.  
Here  $\langle \text{mathstyle} \rangle$  stands for one of the control sequences  $\backslash\text{dispstyle}$ ,  $\backslash\text{textstyle}$ ,  $\backslash\text{scriptstyle}$ ,  $\backslash\text{scriptscriptstyle}$  discussed in Chapter 17. The specified style will apply from this point on, until it is redefined or until the end of the current formula or subformula.  $\text{T}_{\text{E}}\text{X}$  remains in math mode.

- $\backslash\text{eqno}$  Separate a display from its equation number.  
(Allowed only in display math mode.) The current list is converted to a displayed formula and saved away in a safe place;  $\text{T}_{\text{E}}\text{X}$  now switches to non-display math mode. The subsequent  $\langle \text{mlist} \rangle$  will become an equation number, placed at the right of the display as explained in Chapter 19.

- $\backslash\text{x}$  Extension to  $\text{T}_{\text{E}}\text{X}$ .  
The control sequence  $\backslash\text{x}$  allows special actions that might exist in some versions of  $\text{T}_{\text{E}}\text{X}$ . (Such extensions are obtained by loading a separately compiled module with the  $\text{T}_{\text{E}}\text{X}$  system; individual users might have their own special extension modules.)

- $\langle \begin{array}{l} \backslash\text{topmark} \\ \backslash\text{botmark} \end{array} \rangle$  Insert the text of a stored mark.

(Allowed only in  $\backslash\text{output}$  routines.)  $\text{T}_{\text{E}}\text{X}$  inserts the specified mark text into its input; see Chapter 23.

- `\halign(spec){(alignment preamble)\cr(alignment entries)}` Append alignment. This is allowed only in display math mode, and only if there are no formulas being displayed outside of this alignment and no `\eqno`. The behavior is identical to `\halign` when it appears in vertical mode, except that `\dispkip` glue is appended above and below the resulting vertical list.

- $\left\langle \begin{array}{c} \circ \\ \backslash cr \end{array} \right\rangle$  Spurious alignment delimiter.

The symbols  $\circ$  and `\cr` are detected deep inside  $\TeX$ 's scanning mechanism when they occur at the proper nesting level of braces, because they cause  $\TeX$  to start scanning a " $\langle v_j \rangle$ " as explained in Chapter 22. Therefore if these symbols appear in math mode, they are ignored, and you get the error message "There's no `\halign` or `\valign` going on."

- $\left\langle \begin{array}{c} \backslash ENDV \\ \backslash par \end{array} \right\rangle$  "Missing \$ inserted."

An `\ENDV` instruction is inserted automatically by  $\TeX$  at the end of each " $\langle v_j \rangle$ " list of an alignment format. (You can't actually give this control sequence yourself; it only occurs implicitly.) A `\par` token occurs either implicitly, as a result of a blank line in the input, or explicitly. Neither case should happen in math mode, so  $\TeX$  issues an error message and inserts a \$ in an attempt to keep going.

- $\left\langle \begin{array}{c} \backslash def \\ \backslash gdef \end{array} \right\rangle$  `(controlseq)(parameter text){(result text)}` Define a control sequence.

The specified control sequence is defined as described in Chapter 20.  $\TeX$  remains in math mode, and the current list is not affected. You are not allowed to redefine certain control sequences like `\baselineskip` and `\:`, since  $\TeX$  relies on these to control its operations at critical points. Definitions with `\def` disappear at the end of the current formula or subformula; definitions with `\gdef` do not. It is best not to apply both `\def` and `\gdef` to the same control sequence in different parts of a manuscript.

- `(dimenparam)(dimen)` Set a dimension parameter.

Here `(dimenparam)` stands for one of the control sequences `\hsize`, `\vsize`, `\maxdepth`, `\parindent`, `\topbaseline`. The corresponding  $\TeX$  parameter is set equal to the specified dimension;  $\TeX$  remains in math mode, and the current list is not affected. This assignment is "global," it holds even after the end of the current formula. The initial default values of these five parameters are (324, 504, 3, 0, 10) points, respectively.

- `(glueparam)(glue)` Define a glue parameter.

Here `(glueparam)` stands for one of the control sequences `\lineskip`, `\baselineskip`, `\parskip`, `\dispkip`, `\dispakip`, `\dispbkip`, `\topskip`, `\botskip`, `\tabskip`. The corresponding  $\TeX$  parameter is set equal to the specified glue;  $\TeX$  remains in

math mode, and the current list is not affected. This assignment is "local," it will be forgotten at the end of the current formula or subformula; so this construction is of very limited utility in math mode. The initial value for all these types of glue is zero.

- `\chcode⟨number₁⟩+⟨number₂⟩` Define a character interpretation.

The character whose seven-bit code is  $\langle \text{number}_1 \rangle$  is subsequently treated as being of category  $\langle \text{number}_2 \rangle$ , where the category codes are described in Chapter 7. This definition will be local to the current formula or subformula.  $\text{\TeX}$  remains in math mode, and the current list is not affected.

- `\chpar⟨number₁⟩+⟨number₂⟩` Define an integer parameter.

$\text{\TeX}$ 's internal parameter  $\langle \text{number}_1 \rangle$  is set equal to  $\langle \text{number}_2 \rangle$ . See Chapter 25 for a table of the internal parameters. This definition will be local to the current formula or subformula, and any new settings of "binopbreak" and "relbreak" will disappear before  $\text{\TeX}$  uses them in the present formula, so they are best defined *outside* of math mode.  $\text{\TeX}$  remains in math mode, and the current list is not affected.

- `\output{⟨vlist⟩}⟨optional space⟩` Set the output routine.

The specified  $\langle \text{vlist} \rangle$  is stored for later use when pages are output (see Chapter 23).  $\text{\TeX}$  remains in math mode, and the current list is not affected. This assignment is "global," it will hold even after the end of the current formula.

- `\setcount⟨digit⟩⟨optional sign⟩⟨number⟩` Set a specified counter.

One of ten counters, indicated by the specified digit, is set to the specified integer value (see Chapter 23). This assignment is "global," it is not rescinded at the end of the formula.  $\text{\TeX}$  remains in math mode, and the current list is not affected.

- `\advcount⟨digit⟩` Advance the specified counter.

The magnitude of the specified counter is increased by 1.  $\text{\TeX}$  remains in math mode, and the current list is not affected.

- `\count⟨digit⟩` Insert the specified counter.

The specified counter is converted to characters (see Chapter 23) and inserted into the input;  $\text{\TeX}$  will read it in math mode.

- $\left\langle \begin{array}{l} \text{\ifeven}\langle \text{digit} \rangle \\ \text{\if}\langle \text{char}_1 \rangle \langle \text{char}_2 \rangle \end{array} \right\rangle \{ \langle \text{true text} \rangle \} \text{\else} \{ \langle \text{false text} \rangle \}$  Conditional text.

$\text{\TeX}$  reads either the true text or the false text, see Chapter 23.

- `\ddt` Print debugging data.

If bit 4 of the `\trace` parameter is 1,  $\text{\TeX}$  prints out its current activities (the lists and pages it is currently building). Furthermore if bit '40 of the `\trace` parameter is 1,  $\text{\TeX}$  will stop, giving you the chance to insert text on-line.  $\text{\TeX}$  remains in math mode, and the current list is not affected.

- (anything else) “! You can't do that in math mode.”

If anything not listed above appears in math mode, you get an error message.  $\TeX$  ignores the token of input that broke the rules, and remains in math mode; the current list is not affected.

### <27> Recovery from errors

OK, everything you need to know about  $\TeX$  has been explained—unless you happen to be fallible.

If you don't plan to make any errors, don't bother to read this chapter. Otherwise you might find it helpful to make use of some of the ways  $\TeX$  tries to pinpoint bugs in your manuscript.

In the trial runs you did when reading Chapter 6, you learned the general form of error messages, and you also learned the various ways you can respond to  $\TeX$ 's complaints. With practice, you will be able to correct most errors “on line,” as soon as  $\TeX$  has detected them, by inserting and deleting a few things. On the other hand, some errors are more devastating than others; one error might cause some other perfectly valid construction to seem wrong. Furthermore,  $\TeX$  doesn't always diagnose your errors correctly, since it is a rather simple-minded computer program that doesn't readily understand what you have in mind. (In other words, let's face it:  $\TeX$  can get hopelessly confused.)

By looking at the input context that follows an error message, you can often tell what  $\TeX$  will read next if you proceed by hitting (carriage-return). For example, look again at the error message discussed at the end of Chapter 6; it shows that  $\TeX$  is about to read “STORY”, then (since the <argument> will be finished) will come “\hskip 0pt” and so on. Here's another example:

```
! Missing { inserted.
<to be read again>
A
(*) \hjust A
nother example.
```

In this case  $\TeX$  has read the “A” and discovered that a “{” was missing. The missing left brace has been inserted and the “A” will be read again, followed by “nother example.” If you understand what  $\TeX$  has read and is going to read

next, you will be able to make good use of the insertion and deletion options when error messages appear on your terminal, because you'll be able to make corrections before an error propagates.

Here is a complete list of the messages you might get from  $\text{\TeX}$ , presented in alphabetic order for reference purposes. Each message is followed by a brief explanation of the problem, from  $\text{\TeX}$ 's viewpoint, and of any remedial action you might want to take. (See also Appendix I.)

! A box specification was supposed to be here.

$\text{\TeX}$  was expecting to see a  $\langle\text{box}\rangle$  now, based on what it had recently seen (e.g., " $\backslash\text{raise}$ " or " $\backslash\text{save}$ " or " $\backslash\text{leaders}$ "), but what it now sees is not the beginning of a  $\langle\text{box}\rangle$ . (See Chapter 24 or 25 or 26 for the definition of a  $\langle\text{box}\rangle$ .) Proceed, and  $\text{\TeX}$  will forget whatever led it to expect a  $\langle\text{box}\rangle$ .

! Ambiguous; you need another { and }.

You seem to be using  $\backslash\text{over}$  or  $\backslash\text{atop}$  or  $\backslash\text{above}$  or  $\backslash\text{comb}$  more than once in the same formula or subformula. Proceed, and the formula will appear as if the current  $\backslash\text{over}$  (or whatever) weren't there.

! All mixed up, can't continue.

$\text{\TeX}$  is quitting, because it is confused about an alignment that has gone awry.

! Argument of  $\langle\text{control sequence}\rangle$  can't begin with }.

The first character of some argument to the specified macro is }. Proceed, and this } will be ignored.

! Bad font link for large delimiter  $\langle\text{number}\rangle$ .

$\text{\TeX}$  is trying to make a variable-size delimiter, but either you gave it the wrong code number or the font information of the current  $\text{ex}$  font is messed up. Maybe the wrong  $\text{ex}$  font has been selected. Proceed, and the delimiter will be changed to "." (blank).

! Blank space should follow file name.

$\text{\TeX}$  usually continues to read a file name until seeing a blank space, so it may have incorporated part of your input text into the file name. Proceed and you might be lucky.

! Display math should end with \$\$.

$\text{\TeX}$  got to a \$ in display math mode, and it wasn't followed by another \$. If you simply have typed a single dollar sign instead of a double one, proceed and  $\text{\TeX}$  will happily pretend there were two. Otherwise you're probably in deep trouble—

but don't give up yet. (Perhaps you didn't want  $\TeX$  to get into display math mode at all; are you doing an alignment with "\$#\$" in some format, where the entry to be aligned is empty, contrary to the advice in Chapter 22?)

! Double subscript.

You can't apply  $\downarrow$  twice to the same thing. Proceed, and the first subscript will be ignored.

! Double superscript.

You can't apply  $\uparrow$  twice to the same thing. Proceed, and the first superscript will be ignored.

! \else required here.

$\TeX$  is processing conditional code initiated by  $\backslash\text{if}$  or  $\backslash\text{ifeven}$ , and the condition was false, so the (true text) has just been skipped over. But the next token was not  $\backslash\text{else}$ ; perhaps the (true text) contains improper grouping of braces. Proceed, and  $\TeX$  will resume reading the input.

(\end occurred on level (number)).

This message may appear on your terminal just before  $\TeX$  signs off; it warns you that the stated number of  $\{$ 's still is waiting to be matched.

! Extra (something).

There are several messages telling you that your input text contains something "extra"; for example, if your input contains a math formula like " $\$x\}+y\$$ ",  $\TeX$  will say that you have an extra " $\}$ ". Proceed, and  $\TeX$  will ignore what it claims is extra. (If you forget to type " $\backslash\text{cr}$ " in an alignment, you may get the message "Extra  $\emptyset$ ", meaning that there are more tabs than specified in the preamble. Your alignment will probably be messed up and overfull boxes will appear; it's too bad.)

! First use of font must define it.

A font code has appeared for the first time in your manuscript, and it wasn't immediately followed by "=" or "+". (This is a rather serious error—always make it a habit to declare your fonts early in your manuscript.) Insert " $\text{=(font file name)\space}$ " and  $\TeX$  will be able to continue.

! \halign in math mode must be preceded and followed by \$\$.

$\TeX$  has just scanned the " $\}$ " that completes an  $\backslash\text{halign}$  in display math mode. You get this error if a nonempty formula preceded the  $\backslash\text{halign}$  or if the current item of input isn't "\$". Proceed, and  $\TeX$  will continue in display math mode. (Strange things may happen.)

! Illegal font code.

You should always refer to fonts as suggested in Chapter 4; for example, you shouldn't type crazy things like "`\:\hjust`" unless you have redefined the control sequence `\hjust`. Insert the font code you intended, by first typing "i".

! Illegal parameter number in definition of `(controlseq)`.

The result text of the stated definition contains an appearance of `#` that isn't followed by `#` or by the number of a parameter in the parameter text. Proceed, and `TeX` will assume that you meant to type "`##`".

! Illegal unit of measure (pt inserted).

`TeX` is scanning a `(dimen)` (see Chapter 10), but the `(number)` isn't followed by any of the two-letter codes `TeX` knows. Proceed, and `TeX` will assume that "pt" was there.

! Improper code.

You are attempting to use `\chcode` or `\chpar` with an improper `(number1)`. The operation is aborted, but you may proceed.

! Input page ended while scanning def of `(controlseq)`.

The `(parameter text)` or the `(result text)` of a `\def`, or the `(mark list)` of a `\mark`, or the `(vlist)` of an `\output`, has extended beyond the current file page of the input file. This probably means that you forgot a "`}`" in some faraway part of the input manuscript, so it's probably a disaster. Insert a right brace if you want, and proceed if you dare.

! Input page ended while scanning use of `(controlseq)`.

This message has been preceded by a "Runaway argument?" message that shows what `TeX` thinks is the beginning of an argument to a defined control sequence. For some reason, a file page in the input file has ended before the text of that argument has ended. This probably is a serious error, because it has presumably gone undetected for a while. You can try to insert something into the input that will terminate the runaway argument, but you most likely should start over, after fixing the argument so that it terminates where it should. (You probably left out a "`}`".)

! Large delimiter `(number)` should be in mathex font.

You are specifying a `(delim)` by a 9-bit code, but you should have specified either  $c_2 = 0$  or  $c_2 \geq '600$ . Proceed, and the delimiter will be selected using  $c_1$  only. (See Chapter 18 for the meaning of  $c_1$  and  $c_2$ .)

! Italic correction must follow an explicit character.  
 The control sequence `\/` is supposed to follow a character from some font, but your input tells `TeX` to apply an "italic correction" to something else. Perhaps you are using a defined control sequence that slants one of its arguments (e.g., `\algbegin` in Appendix E), where the argument ends with a math formula instead of a word. Proceed.

! Limit switch must follow math operator.  
 If the control sequence `\limitswitch` doesn't follow an Op box, it doesn't accomplish anything. Proceed.

! Lookup failed on file `(filename)`.  
`TeX` can't find the file you indicated. Type "i" and insert the correct file name (followed by a blank space and `(carriage-return)`). But be careful: You get only one more chance to get the file name right, otherwise `TeX` will decide not to input any file just now.

! Missing `(something)` inserted.  
 This message can arise in lots of ways and it can name a variety of things that `TeX` sometimes thinks are missing. For example, if you type

$$\left(x+\right)$$

in math mode, `TeX` thinks (correctly) that there's a missing `)`. (See Chapter 26.) In general, when you get this message, `TeX` has already inserted what it says was missing—don't insert another one. If `TeX` has guessed correctly, just proceed. Otherwise, it may be fun to try getting `TeX` back into synch; you might get the message "Missing `)` inserted" followed by one that says "Too many `)`'s", indicating a certain lack of logic on `TeX`'s part.

! Missing digit (0 to 9), 0 inserted.  
`TeX` was expecting to see a decimal digit following `\box` or `\save`, but it isn't there. Proceed; `TeX` has already inserted a "0".

! OK.  
 This isn't an error message. `TeX` is stopping because you asked it to `(\ddt` with `\trace bit '40` set).

! Only one `#` allowed per tab.  
 A `(format)` in an alignment preamble must have exactly one `#`, but you seem to have typed more than one. Proceed, and the extra `#` will be ignored.



! Only single characters can be accented in horizontal mode. An `<accent>` has not been followed by a proper `<accentec>`. Proceed, and the `<accent>` will be ignored.

! `\output` routine didn't use `\page`.  
A page was assembled for output, but the `\output` routine didn't make use of it, so it is lost forever. Proceed.

! Parameters must be numbered consecutively.  
You must say #1, #2, etc., in order, when designating parameters in the `<parameter text>` of a macro definition. When you get this message, `TeX` has already inserted the correct parameter number, so you may want to delete an incorrect one before proceeding.

Overfull box, . . .

This is an information message, not an error message (i.e., `TeX` doesn't stop). The box whose contents are partially displayed is "overfull" because it doesn't have enough glue shrinkage to get down to the required size. Thus the box contents are too wide or too high by the specified amount; in your output you will probably see this box sticking out somewhere or overlapping another one, unless the excess is very small. Overfull boxes can arise from a variety of reasons, notably when there is no decent way to break the lines of certain paragraphs, or when a displayed equation is too wide to fit on a single line. You may want to settle for badly broken lines in a paragraph, by increasing the value of `\jpar` as discussed in Chapter 14; or you might be able to help by inserting discretionary hyphens, especially if there is a word that `TeX` doesn't try to hyphenate (e.g., "Inter\~change" in the first line of Appendix F). But in a high-quality job an overfull box usually means that the author should rewrite the text, eliminating the problem entirely.

Runaway argument?

This message is followed by the tokens of a macro argument that didn't end where you wanted it to. (See "!" Input page ended while scanning use of . . ." above.)

! `TEX capacity exceeded, sorry [(size)=(number)]`.

This is a bad one. Somehow you have stretched `TeX` beyond its finite limits. The thing that overflowed is indicated in brackets, together with its numerical value in the `TeX` implementation you are using. The following table shows the internal

sizes that might have been exceeded:

<code>alignsize</code>	the number of simultaneous alignments;
<code>fmemsize</code>	the number of words of auxiliary font information;
<code>hashsize</code>	the number of different multiletter control sequences;
<code>idlevs</code>	logarithm of the number of levels of grouping;
<code>memsize</code>	memory used to store tokens and many other types of things;
<code>nestsize</code>	number of simultaneous partially-complete lists;
<code>parsize</code>	number of simultaneous partially-scanned arguments;
<code>savesize</code>	number of values to restore at end of group or formula;
<code>stacksize</code>	number of simultaneous levels of input;
<code>stringsize</code>	number of independent operations on typesetting device;
<code>varsize</code>	memory used to store boxes and many other types of things.

If your job is error-free, the remedy is to recompile the  $\TeX$  system, increasing what overflowed. However, there's probably something you can do to your job that will make it run. Maybe you have specified an infinite macro-expansion; then it would cause overflow no matter how big you make  $\TeX$ . If `savesize` has overflowed, you probably have started a group and forgot to finish it. (Every time you change fonts, say, inside a group, an entry is being saved, *unless* you are on level zero.) Or perhaps you are trying to specify a gigantic alignment that spans more than a page;  $\TeX$  has to read all the way to the end of an alignment before outputting any of it, so this consumes huge amounts of memory space. (It's necessary to limit your alignments to reasonable size, by using a fixed format for the multipage cases.) As the message says, it is a sorry situation.

! There's no `\halign` or `\valign` going on.

Your input contains a `@` or a `\cr` that didn't get recognized as part of an alignment, perhaps because you didn't mean to type it, but most likely because some alignment entry doesn't have properly-balanced grouping.  $\TeX$  has deleted the offending `@` or `\cr`; to recover, try to insert braces that balance the group, followed by the current token. For example, if your input was "`{x@`" up to this point, the "`{`" is hiding the "`@`"; type "`i`" and then insert "`}@`".

! This can't happen.

Something really unexpected has caused  $\TeX$  to come to a screeching halt.

! This is allowed only in output routines.

The current input token will be ignored, since it specifies an operation not available except when  $\TeX$  is running an `\output` routine (and  $\TeX$  isn't).

! This dimension shouldn't be negative.

You were naughty and tried to specify a negative `(dimen)` where it isn't allowed. Proceed, and the dimension will be assumed zero.

! Too many `}`'s.

You are not inside a group, so the `}` just scanned will be discarded when you proceed.

! Too much stretch for proper line breaking.

This message usually occurs when you're doing something like `"\hjust to (dimen){...}"` and TeX's line-breaking procedure was invoked, to break an overfull box into several lines as described in Chapter 21. In such cases, `"\hfill"` shouldn't be used in the box; TeX will not break lines in a paragraph when the glue has more than one million points of accumulated stretchability. (The reason for this is that the computations are performed with limited-precision arithmetic, and the spacing will come out looking bad if TeX tries to make precise measurements after subtracting infinity from infinity.) Proceed, and you'll probably see how bad it looks.

! Undefined control sequence.

TeX has encountered a control sequence it doesn't know; see Chapters 24, 25, or 26 for hints on how to fix this.

! Unknown delimiter.

The `(delim)` you have specified isn't one of those listed in Chapter 18. Proceed, and TeX will use a blank delimiter.

! Use of `(controlseq)` doesn't match its definition.

You have typed something that doesn't follow the rules of the specified control sequence. (For example, consider the control sequence `\ansno` of Appendix E. If you type `"\ansno 5.No."`, you have forgotten the space that's required after `"5."`.) TeX will proceed by assuming that the thing you typed was the thing that was required; thus, in the above example, TeX will assume that the `"N"` is a space, and your best strategy is to insert a new `"N"`.

! Whoa---you have to define a font first.

TeX has aborted your job, because it can't do what you asked it to do without having some font selected as the "current font."

! You can only define a control sequence.


Your manuscript apparently contains `\def` or `\gdef` and the next thing wasn't a control sequence. Proceed, and TeX forgets that the `\def` or `\gdef` occurred.

For example, if you typed `"\def ansno"` when you meant `"\def\ansno"`, TeX will read the "a" and complain; to recover, you should delete the next four tokens (namely "ansno"), then insert `"\def\ansno"`.


! You can't do that in (mode).

Your manuscript is trying to do something incompatible with TeX's current mode. TeX will ignore the token it has just read; so the proper way to recover is usually to insert something that takes TeX into the correct mode (e.g., `"\par"` will usually go from horizontal mode to vertical mode, and `"$"` will usually go from math mode to horizontal mode), followed by the current token again. For example, suppose you have typed `"\vskip .5 in"` before ending a paragraph; TeX will stop before it reads the ".5". To recover, type "i" for insertion, then type `"\par\vskip"` and (carriage-return).


! You can't redefine this control sequence.  
You have discovered one of TeX's reserved control sequences. The `\def` or `\gdef` will be ignored if you proceed.

 Exercise 27.1: What is the best way to recover from the following error?

```
! You can't do that in math mode.
\sl →\:
      n
p.3,1.307 $x+y is {\sl
                                not} zero.
```

 Exercise 27.2: And what about this one?

```
! Illegal units of measure.
<to be read again>
                                P
<to be read again> p
                                {
(*) \hjust to 5Op{Test}
```

 You can get more information from TeX if you make use of its tracing capability. Type `"\trace`mmmnnnxy"` (using the control sequence `\trace` defined in Appendix B) to set up the combination of tracing facilities you want, according to the

following cryptic encoding scheme:

*mmm* is an octal code for the number of items per list that will be shown when a box is displayed. (If  $mmm = 0$ , it is automatically changed to 5.)

*nnn* is an octal code for the number of levels deep that will be shown when a box is displayed.

*x* equals (1, if you want to see what replacements are being made in macros as they are expanded)

plus (2, if you want each line of your input files to be entered on your terminal before it is processed by  $\TeX$ , giving you a chance to edit it first)

plus (4, if you want  $\TeX$  to stop whenever the control sequence `\ddt` appears in the input).

*y* equals (1, if you want to be told about "overfull boxes")

plus (2, if you want to see the gory details about what is being typeset on each page before it is shipped to the `\output` routine)

plus (4, if you want to see  $\TeX$ 's current activities whenever the control sequence `\ddt` appears in the input).

The normal setting is `\trace^345`. Thus  $\TeX$  normally shows boxes to depth 3, with up to 5 items per list; it stops and dumps on `\ddt` calls; and it shows overfull boxes. If you say "`\trace0`" you get *nothing*, while if you say "`\trace^77777777`" you probably get *too much*. Boxes are displayed when they are overfull, or when they are completed pages, or when they are in the list of current activities, but only if the current *x* or *y* setting calls for information about these boxes. The contents will appear on your terminal as well as on the "errors.tmp" file; and the format of this information is self-explanatory, once you understand it. You can, of course, change the combination of tracing facilities as many times as you want to, so that you aren't deluged with information when you don't want any.

Final hint: When working on a long manuscript, it's best to prepare only a few pages at a time. Set up a "galley" file and a "book" file, and enter your text on the galley file. (Put control information that sets up your basic format and fonts at the beginning of this file, so that you don't have to retype it each time.) After the galleys come out looking right, you can append them to the book file; then you can run the book file through  $\TeX$  once a week, or so, in order to see how the pages really fit together. For example, when the author prepared this

manual, he did one chapter at a time; and Chapter 18 was split into three parts, because of its incredible length.

Final exhortation: GO FORTH now and create *masterpieces of the publishing art!*

### <A> Answers to all the exercises

2.1: ```$\,$`` or ```\2`` (but not ``````); ``{}```` or `{}````, etc.

3.1: `math\`ematique`, `math\`ematique`; `centim\`etre`.

4.1: Ulrich Dieter, `{\sl Journal f\"ur die reine und angewandte Mathematik \bf 201}` (1959), 37--70. (Note in particular the use of "--" to get an en-dash, did you remember that?)

5.1: Type `{-}-` or `--{-}` or `{-}{-}` or `-{-}`, etc.

5.2: No—the first definition is pretty lousy because it accomplishes nothing! (When `\rm` appears in the subsequent text it will be replaced by `{\:a}`, but this font change immediately disappears because it's inside a group.)

5.3: It could end with any character that has been `\chcoded` to 2 at the time the group ends. After that point the effect of all `\chcodes` inside the group will be lost.

6.1: Type "i" (for insert). Then when  $\TeX$  prompts you for more input, type `"\c.c"`; this will be inserted at the current place in the input (the undefined `\cc` has already been discarded), and then  $\TeX$  resumes reading the original line (i.e., it will then read the comma; you shouldn't insert another comma, since the comma wasn't in error).

7.1: Yes, if the format you are using (e.g., `basic`) has defined `%` to be an end-of-line character (type 5).

9.1: `{\sl Commentarii Academ\ao\ Petropolitan\ae}` is now `{\sl Doklady Akademi\t\i a Nauk SSSR}`.

9.2: `\O ystein Dre`, `\t IUri \t IAnov`, `Ja`far al-Khow\A arizm\A\i`, and `W\l ladyis\l law S\"u\ss man`.

10.1: Here is one of many possible solutions.

```
\def\1{\hjust to 5mm{\hfill\vrule depth 4pt}}
\def\2{\hjust to 5mm{\hfill\vrule depth 8pt}}
\vjust{\hrule\hjust{\vrule depth 8pt
\1\2\1\2\1\2\1\2\1\2\1\2\1\2\1\2\1\2\1\2}}
```

12.1: 25, 41, and 12 units, respectively.

12.2: "... launched by \hjust{NASA}."; or "... launched by NASA\null."

16.1:  $\$2^{n+1}$ ,  $\$(n+1)^2$ ,  $\$\sqrt{1-x^2}$ ,  $\$\overline{w+\overline{z}}$ ,  $\$p_{1+e+1}$ ,  $\$a_{b+c+d+e}$ ,  $\$f_{\prime}(x)$ .

16.2: No space will be typeset after the "f". (Also, it would have been slightly better to end with "\$y\$".)

16.3: Deleting an element from an  $n$ -tuple leaves an  $(n-1)$ -tuple.

17.1:  $\$\binom{p}{2}x^2y^{p-2} - \frac{1}{1-x}\frac{1}{1-x^2}$ .

17.2:  $\$\sum_{i=1}^p\sum_{j=1}^q\sum_{k=1}^r a_{ij}b_{jk}c_{ki}$ .

17.3:  $\$\sum_{\{1\leq i\leq p\}\atop\{1\leq j\leq q\}\atop\{1\leq k\leq r\}} a_{ij}b_{jk}c_{ki}$ .

18.1:  $\$n\text{\hjust{:d th}}$ root$ .

18.2:  $\$\biglpx-s(x)\biglpy-s(y)\biglpy$ . (Note that the period is included in this display.)

18.3:  $\$\frac{1}{2\pi}\int_{-\infty}^{\infty}\sqrt{y}\sum_{k=1}^n\sin 2x+k(t)\biglrf(t)+g(t)\biglrdt$ .

18.4:  $\$\frac{(n+1+n+2+\dots+n+m)!}{n+1!\cdot n+2!\cdot\dots\cdot n+m!} = \binom{n+1+n+2}{n+2}\binom{n+1+n+2+n+3}{n+3}\dots\binom{n+1+n+2+\dots+n+m}{n+m}$ .

18.5:  $\$\left(\text{\cpile{y+1\cr\vdots\cr y+k\cr}}\right)$ .

18.6:  $\$\Psiscr{Lh}(x) = \text{\hjust{Tr}\left[\frac{\partial F_{L+1}}{\partial t_{jh}}\chi(L)\Mscr{n}(x)\right], \quad \text{\hjust{evaluated at}}\chi(\Gamma)\text{\modop\hjust{\sl SL}(n, \text{\hjust{\bf C}})}$ .

(Here "\hjust{\sl SL}" gives slightly better spacing than simply SL, because it suppresses the italic correction on the S.)

18.7:  $\$\def\o{\mathrel{(:)=}}$ . (The braces prevent space between : and =, since they specify one-character subformulas that are converted into Ord boxes.) Another solution is  $\$\def\o{\mathrel{\char"72\char"75}}$ .

20.1: The ## feature is indispensable when the result text of a definition contains *other* definitions. (We will see later that ## is also useful for alignments; cf. the definitions of \eqalign and eqalignno in Appendix B.)

21.1: When a null box was placed on the vertical list below the "s" box, the \lineskip glue of 3 points was not inserted, because the \baselineskip distance of 0 points was not exceeded. Thus the interline glue was computed to be 0 points, and the blank line didn't show up.

21.2: `\vjust{\baselineskip-1pt\lineskip 3pt\halign{\ctr{#}\cr T\cr h\cr i\cr s\cr \cr b\cr o\cr x\cr}}` .

21.3: The width-so-far minus glue-shrink-so-far must never exceed the final desired size of 0 points, otherwise the `\hjust` will go into an unwanted excursion into paragraph line-breaking. (Thus the argument to `\spose` had better not be more than 100 points wide.)

21.4: `\def\boxit#1{\vjust{\hrule\hjust{\vrule\hskip 3pt \vjust{\vskip 3pt #1 \vskip 3pt}\hskip 3pt\vrule}\hrule}}` .

21.5: `\leaders\chop to 0pt{\hjust to size{\hfill*}\lower 3.75pt \hjust{*}*}\lower 3.75pt\hjust{*}*}\vfill` . [For more interesting effects, try `\leaders` inside of boxes used as leaders.]

22.1: The equation number "(13)" would appear on the bottom line instead of being centered vertically. (A box constructed by `\vjust` has the same baseline as the bottom box in the vertical list.)

23.1: Since the `\output` routine might occur at an unpredictable time, the value of `\hsize` may not be 4.5 inches. (On the other hand, if it is known that the manuscript never diddles with `\hsize`, the output routine will be more readable if `\ctrline` is used for the running title and page number lines.)

23.2: For example, you can replace it by the following:

```
\output{\vjust to 7.5in{\baselineskipOpt\lineskipOpt
  \if T\tpage{\vskip .5in}
  \else{\vjust to .15in{\vfill
    \def\lead{\leaders\hrule\hfill}
    \hjust to 4.5in{\ifevenO{\countO\lead\ :b\topmark}
      \else{\ :b\topmark\lead\cp
        \vskip .35in}
    \page\vfill
    \if T\tpage{\gdef\tpage{F}
      \hjust to 4.5in{\hfill\ :c\countO\hfill}}
    \else{}}\advcountO} .
```

27.1: (A "\$" was forgotten after the "y".) If you just insert a dollar sign now, the "{" will be unmatched in the math formula, so TeX will stop again after inserting a "}" before the "\$" you just inserted; this will cause unbalance and possible embarrassment. The correct procedure is to insert "}\$\:", then TeX will proceed almost as if the error hadn't happened.



27.2: T<sub>E</sub>X has already decided that "pt" was intended but missing from the input. If you simply proceed now, T<sub>E</sub>X will insert a "{" and give you another error message (after which you'll have to delete "p" and "{"). The correct procedure is to delete the "p" now (by typing "1"); then type  $\langle$ carriage-return $\rangle$ . The error has been fully corrected (unless picas were meant instead of points).

### <B> Basic T<sub>E</sub>X format

The following listing of file "basic.TEX" shows how to give T<sub>E</sub>X enough knowledge to do the "basic" things mentioned in the main text.

```

\chcode`173+1 \chcode`176+2 \chcode`44+3 \chcode`26+4
\chcode`45+5 \chcode`43+6 \chcode`136+7 \chcode 1+8
\def\%{\char`45 } % Note, the space after 45 is needed! (e.g.\%0)
\def\lft#1{#1\hfill}
\def\ctr#1{\hfill#1\hfill}
\def\rt#1{\hfill#1}
\def\rjustline#1{\hjust to size{
  \hskipOpt plus1000cm minus1000cm #1}}
\def\ctrline#1{\hjust to size{\hskipOpt plus1000cm minus1000cm
  #1\hskipOpt plus1000cm minus1000cm}}
\def\trace{\chpar0+} \def\jpar{\chpar1+} \def\ragged{\chpar8+}
\def\log{\mathop{\char`154\char`157\char`147}\limitswitch}
\def\lg{\mathop{\char`154\char`147}\limitswitch}
\def\ln{\mathop{\char`154\char`156}\limitswitch}
\def\lim{\mathop{\char`154\char`151\char`155}}
\def\limsup{\mathop{\char`154\char`151\char`155
  \,\char`163\char`165\char`160}}
\def\liminf{\mathop{\char`154\char`151\char`155
  \,\char`151\char`156\char`146}}
\def\sin{\mathop{\char`163\char`151\char`156}\limitswitch}
\def\cos{\mathop{\char`143\char`157\char`163}\limitswitch}
\def\tan{\mathop{\char`164\char`141\char`156}\limitswitch}
\def\cot{\mathop{\char`143\char`157\char`164}\limitswitch}
\def\sec{\mathop{\char`163\char`145\char`143}\limitswitch}
\def\csc{\mathop{\char`143\char`163\char`143}\limitswitch}
\def\max{\mathop{\char`155\char`141\char`170}}
\def\min{\mathop{\char`155\char`151\char`156}}

```

```

\def\sup{\mathop{\char`163\char`165\char`160}}
\def\inf{\mathop{\char`151\char`156\char`146}}
\def\det{\mathop{\char`144\char`145\char`164}}
\def\exp{\mathop{\char`145\char`170\char`160}\limitswitch}
\def\Pr{\mathop{\char`120\char`162}}
\def\gcd{\mathop{\char`147\char`143\char`144}}
\def\choose{\comb()}
\def\leftset{\mathopen{\{\,}}
\def\rightset{\mathclose{\,\}}
\def\modop{\langle\, \mathbin{\char`155\char`157\char`144}\penalty900\langle\,}
\def\mod#1{\penalty0\langle\char`155\char`157\char`144\,\, #1}
\def\eqv{\mathrel\char`421}
\def\neqv{\mathrel{\not\eqv}}
\def\qqquad{\quad\quad}
\def\ldots{{. \. \.}}
\def\cdots{{\char`401\char`401\char`401}}
\def\ldotss{{. \. \. \.}}
\def\cdotss{\cdots}
\def\ldotssm{{\. \. \. \.}}
\def\vdots{\vjust{\baselineskip 4pt\vskip 6pt
\hjust{.}\hjust{.}\hjust{.}}}
\def\equalign#1{\vcenter{\halign{\hfill$\dispstyle{##}$\!
@$\dispstyle{\null##}$\hfill\cr#1}}}
\def\equalignno#1{\vjust{\tabskipOpt plus1000pt minus1000pt
\halign to size{\hfill$\dispstyle{##}$\tabskip Opt
@$\dispstyle{\null##}$\hfill
\tabskipOpt plus1000pt minus1000pt
@$\hfill##$\tabskip Opt\cr#1}}}
\def\cpile#1{\vcenter{\halign{\hfill##\hfill$\cr#1}}}
\def\lpile#1{\vcenter{\halign{##\hfill$\cr#1}}}
\def\rpile#1{\vcenter{\halign{\hfill##$\cr#1}}}
\def\null{\hjust{}}
\def\twolines#1#2#3{\halign{\hjust to size{##}\cr$\quad\dispstyle
{#1}$\hfill\cr\noalign{\penalty1000\vskip#2}
\hfill$\dispstyle{#3}\quad$\cr}}
\def\chop to#1pt#2{\hjust{\lower#1pt\null\vjust{\hjust{\lower99pt
\hjust{\raise99pt\hjust{\dispstyle{#2}$}}}\vskip-99pt}}}
\def\spose#1{\hjust to Opt{\hskipOpt minus100pt
#1\hskipOpt minus1000000pt}}

```

```

\:\@+cmathx
\:\a+cmr10 \:\d+cmr7 \:\f+cmr5
\:\g+cmi10 \:\j+cmi7 \:\l+cmi5
\:\n+cms10
\:\q+cmb10
\:\u+cmsy10 \:\x+cmsy7 \:\z+cmsy5
\:\?+cmt110

\def\rm{\:\a} \def\sl{\:\n} \def\bf{\:\q} \def\it{\:\?}

\parindent 20pt \maxdepth 2pt \topbaseline 10pt
\parskip 0pt plus 1 pt \baselineskip 12pt \lineskip 1pt
\dispSKIP 12pt plus 3pt minus 9pt
\dispasKip 0pt plus 3pt \dispbskip 7pt plus 3pt minus 4pt

\def\biglp{\mathopen{\vcenter{\hjust{\:\@char`0}}}}
\def\bigrp{\mathclose{\vcenter{\hjust{\:\@char`1}}}}
\def\bigglp{\mathopen{\vcenter{\hjust{\:\@char`22}}}}
\def\biggrp{\mathclose{\vcenter{\hjust{\:\@char`23}}}}
\def\biggglp{\mathopen{\vcenter{\hjust{\:\@char`40}}}}
\def\biggggrp{\mathclose{\vcenter{\hjust{\:\@char`41}}}}

\mathrm adf \mathit gjl \mathsy uxz \mathex @

\output{\baselineskip20pt\pagectrline{\:\a\count0}\advcount0}
\setcount0 1

\rm
\null\vskip-12pt % allow glue at top of first page

```

**<E> Example of a book format**

This appendix contains two parts: First comes a supplement to the  $\text{\TeX}$  report, explaining the main conventions a typist uses when entering material from *The Art of Computer Programming (ACP)* into the system. Second is a listing of file `acphdr.TEX`, in which the precise format for those books is defined in terms of  $\text{\TeX}$  control sequences. As you read the first part of this appendix, try to imagine that you yourself are a typist with the responsibility for inputting part of the manuscript for this series of books.

Several examples below are best understood if you have a copy of *ACP* handy; so why not go fetch your copy of Volume 1 now? (And if you have Volume 2, that will help even more.)

- Since this appendix must cover a wide range of topics in a reasonably short space, it is rather terse; please forgive the author for this. Every time you see “•” in this appendix, you’re being hit with a new topic.

- Everything in Appendix B—the “basic” format that is explained throughout the user manual—is used also in *ACP*, except that the conventions for number theory are slightly different. (See Chapter 18, part 8, for a discussion of Appendix B’s approach to number theory.) To typeset “ $z \equiv 0 \pmod{pq}$ ”, type “`$x \equiv 0 \pmod{pq}$`”; and to typeset the operator “mod” you can use `\mod` instead of `\modop`. There also is one further control sequence defined for mathematics, namely `\deg` for the degree symbol: type “`$45\deg$`” to get “45°”.

- The style of typical bibliographic references is “`{\author name}, {\s1{name of book or journal}}{\bf{volume number}}{(year)}, {starting page}--(ending page).`” For example,

`M. R. Garey et. \ al., {\s1 SIAM J. Appl. \ Math. \ \bf34} (1978), 477--495.`

Another example appears in the answer to exercise 4.1 (see Appendix A).

- Remember to type “\” after any abbreviation in which a lower case letter is followed by a period followed by a space, when this period is not the end of a sentence. Abbreviations aren’t used very much in *ACP*, but they do occur frequently in bibliographic references (as in the example just given). Furthermore you should be on the lookout for the following commonly-used abbreviations:

`Eq. \ Eqs. \ Fig. \ Figs. \ cf. \ ed. \ etc. \`

The special abbreviations “A.D.” and “B.C.”, sometimes used in dates, are typed “`{\ :m A.D.}`” and “`{\ :m B.C.}`”, respectively, in order to get them into the small caps font.

• Remember to type en-dashes, not only when giving page numbers in bibliographic references but also in constructions like the following:

exercise 3.1--6      Table 3.2.1.1--1      Fig. \ A--1

• Each major section of *ACP* starts on a new page. (A major section is a section whose number contains just one decimal point, for example "Section 3.2".) A separate computer file is maintained for each major section; for example, file `v232.TEX` contains Volume 2, Section 3.2. Such a file starts out with the following fixed information:

```
\input acphdr
\runninglefthead{(chapter title with all letters capitalized)}
\titlepage\tenpoint
\vflll
\ctrline{SECTION (major section number) OF
        THE ART OF COMPUTER PROGRAMMING}
\ctrline{$\copyright$ (year)
        Addison--Wesley Publishing Company, Inc.}
\vflll
\runningrighthead{(section title with all letters capitalized)}
\section{(major section number)}
\eject\setcount0 (starting page number)
\sectionbegin{(major section number). \ (section title with all letters capitalized)}
```

For example, the last four lines of this introductory information have the following form on file `v232.TEX`:

```
\runningrighthead{GENERATING UNIFORM RANDOM NUMBERS}
\section{3.2}
\eject\setcount0 9
\sectionbegin{3.2. GENERATING UNIFORM RANDOM NUMBERS}
```

The beginning of a major section is a major event in *ACP*, so you are asked to type all of the above—no special control sequence has been made for it.\*

One further piece of fanciness is used at the beginning of a major section: The first words of the opening sentence are typeset with capital letters from font `\:c` in place

---

\*The beginning of a chapter is an even more major event; the format for such a gala occasion won't be described here, since the author will do the first page of each chapter by himself, just to keep his hand in.

of lower case letters. For example, the four lines that we have quoted from `v232.TEX` are immediately followed in that file by

```
I{\:cN THIS SECTION} we shall consider methods
```

(and the result when typeset looks like this: "IN THIS SECTION we shall consider methods").

- A minor section of ACP is one whose number contains two decimal points, for example "Section 4.2.2". Each minor section starts out with four special lines

```
\runningrighthead{(section title with all letters capitalized)}
\section{(minor section number)}
\sectionskip
\sectionbegin{(minor section number).□(section title partially capitalized)}
```

followed by the text of the first paragraph. "Partially capitalized" means that you capitalize only major words, as in the title of a book. For example:

```
\runningrighthead{ACCURACY OF FLOATING-POINT ARITHMETIC}
\section{4.2.2}
\sectionskip
\sectionbegin{4.2.2. Accuracy of Floating-Point Arithmetic}
Floating-point computation is by nature inexact, and ...
```

Thus, a minor section has much less fanfare, and there is no messing around with font `\:c`.

- A diminished section of ACP is one whose number contains three decimal points, for example "Section 1.2.11.1". This is typed just the same as a minor section, except that you omit the `\sectionskip`, you use `\dimsectionbegin` instead of `\sectionbegin`, and you capitalize only the first word of the section title. For example:

```
\runningrighthead{THE O-NOTATION}
\section{1.2.11.1}
\dimsectionbegin{\star 1.2.11.1. The $O$-notation}
A very convenient notation for dealing with ...
```

This example illustrates another thing: you type "`\star`" just after "`sectionbegin{`" when beginning a "starred" section or subsection. (TEX will then insert an asterisk in the left margin.) Such stars occur sometimes even in major sections.

• A subsection of ACP ranks lowest in the hierarchy. It is part of a section that is introduced by a bold-face subhead, but this subhead never gets into the running headline at the top of right-hand pages. You specify the beginning of a subsection simply by typing

```
\subsectionbegin{<subhead>}
```

followed by the opening paragraph of the subsection. Don't type a period after the subhead— $\TeX$  will typeset one anyway, it's part of the subsection format—and if you include another period there will be two! This is consistent with the titles of sections in general (see the examples above); you never put a period before the }.

Here are two examples of subsection format, taken from within sections 1.3.3 and 3.3.2 of ACP:

```
\subsectionbegin{Products of permutations}
We can ``multiply`` two permutations together, ...
\subsectionbegin{E. Coupon collector's test}
This test is related to the poker test ...
```

• Special events like theorems and algorithms sometimes occur in the text of a section, and they have their own special format. Type

```
\algbegin <name of algorithm or program>□(<descriptive title>) .□
```

at the beginning of an algorithm or program. For example (taken from pages 2 and 141 of Volume 1):

```
\algbegin Algorithm E (Euclid's algorithm). Given two ...
\algbegin Program M (Find the maximum). Register assignments: ...
```

Similarly, you type

```
\thbegin <name of theorem or lemma or corollary> .□
```

at the beginning of a theorem or lemma or corollary. The text of a theorem or lemma or corollary is set in *slanted* type, with any embedded math formulas set off by \$'s as usual (so that italic letters will be distinguishable from slanted ones). For example,

```
\thbegin Corollary P. {\sl If a  $[0,1]$  sequence is
 $k$ -distributed, it satisfies the permutation test of
order  $k$ , in the sense of Eq. (10).}
```

Be sure to remember the final } that turns off the \sl, otherwise you'll see a lot of slantedness in the following text.

- When beginning the proof of a theorem, type “\proofbegin” (with no period following it) instead of “Proof.”. For example,

```
\proofbegin It is clear that ... .
```

(But use \dproofbegin if the preceding paragraph ended with a display.) At the end of the last paragraph of a proof, type the following ritual:

```
\quad\blackslug
(empty line to end the paragraph)
\yyskip
```

This typesets a “□” and leaves extra space before the paragraph that follows. The same ritual is used also at the end of the last step of an algorithm.

- Speaking of the steps of algorithms, each step is a separate paragraph. At the beginning of that paragraph the instructions

```
\algstep (step number). □[(description of step)]
```

should be typed. For example, the following comes from page 2 of Volume 1:

```
\algstep E1. [Find remainder.] Divide  $m$  by  $n$  and let  $r$ 
be the remainder.\xskip (We will have  $0 \leq r < n$ .)
```

```
\algstep E2. [Is it zero?] If  $r=0$ , the algorithm
terminates;  $n$  is the answer.
```

```
\algstep E3. [Interchange.] Set  $m+n$ ,  $n+r$ , and go back
to step E1.\quad\blackslug
```

```
\yyskip Of course, Euclid did not present his algorithm in
just this manner. The above format illustrates the style
in which all of the algorithms throughout this book will
be presented.
```

- Within a paragraph, type “\xskip” before and after parenthesized sentences. (For example, there is an \xskip in the paragraph you're now reading, and in algstep E1 above.)



• Sometimes the author wants to insert extra space between paragraphs of a section, in order to indicate a slight change of topic. For this you type “\yskip” just before the new paragraph. (The space corresponding to \yskip turns out to be just half the space corresponding to \yyskip.)

Another use of \yskip sometimes occurs when paragraphs appear in series, with “a)” inserted in place of the indentation in the first paragraph, “b)” in the next, and so on. For this you type “\yskip\textindent{a)}”. Also add the control sequence “\hang” if the entire paragraph (except for the “a)”) is to be indented. For example, the paragraph you are about to read next has been typeset with the instructions

```
\yskip\textindent{${\bullet}$}Sections normally end ...
```

• Sections normally end with a group of exercises. At this point you type

```
\xbegin{EXERCISES}
```

or (in some cases) “\xbegin{EXERCISES---First Set}”, etc. Then come the exercises, one by one, each starting a new paragraph. At the beginning of this paragraph you type

```
either \exno <number> .□[<rating>]
or \trexno <number> .□[<rating>]
```

where \trexno is used if the exercise is supposed to have a triangle in the margin. For example,

```
\exno 4. [M50] Prove that when $n$ is an integer, $n>2$, the
equation $x^n+y^n=z^n$ has no solution in positive integers
$x$, $y$, $z$.
```

After the “[<rating>]” of an exercise there sometimes is a parenthesized descriptive title, or the name of the originator of the exercise. The descriptive title, if present, should be slanted; names should not. The closing right parenthesis should be preceded by a period and followed immediately by “\xskip” without any intervening space. For example (see ACP Volume 1, page 20):

```
\exno 14. [50] (R. W. Floyd.)\xskip Prepare a computer program ...
\trexno 15. [HM28] ({\sl Generalized induction.})\xskip The ...
```

If the exercise contains subparts (a), (b), etc., there are two cases: The subparts may be introduced by \textindents (as in the exercise 15 we were just looking at on

page 20 of Volume 1), or they may be embedded in a paragraph (as in exercise 29 on page 26). The first case should be treated by making separate paragraphs introduced by “\hang\textindent{a}”]; put \xskip before the first such paragraph, but not before the others. In the second case, type “\xskip (a)” and “\xskip (b)”, etc., where there is no space before the \xskip.

If the exercise contains a “hint” within a paragraph, you type “\xskip[{\s1 Hint:}”]; as usual, there should be no space before \xskip.

• Answers to the exercises appear at the back of the book; they are entered on a separate file—e.g., v2ans.TEX for the answers of Volume 2. It is best to typeset the answers for each individual section at the same time as you typeset the exercises for that section, in order to ensure consistency. In the answer pages you say

```
\ansbegin{(section number)}
```

just before the answers to the exercises for a particular section. Then each answer is preceded by

```
\ansno (number).␣
```

For example (reading from Volume 1, page 465),

```
\ansbegin{1.1}
```

```
\ansno 1.  $t+a$ ,  $a+b$ ,  $b+c$ ,  $c+d$ ,  $d+t$ .
```

```
\ansno 2. After the first time, ...
```

Now look at answer number 3 on that page of Volume 1; here you should *not* use “\algbegin”, since \algbegin is for algorithms in the text. By looking at the formal definition of \algbegin in the later part of this appendix, you can see how to modify it in order to handle this particular case, namely to type

```
\ansno 3. {\bf Algorithm F }({\s1 Euclid's  
algorithm\}){\bf.}\xskip Given two positive ...
```

In still more complicated cases you may have to typeset the exercise number yourself in connection with \halign. Then you use \anskip just before the answer, in order to get the proper spacing between answers.

Sometimes one answer is given for two or more exercises. In this case you use “\ansnos” instead of “\ansno”. For example (please turn to page 599 of Volume 1),

```
\ansnos 15, 16.  $rI1 \equiv \{P0\}$ ,  $rI2 \equiv \{P1\}$ , ...
```

• This last example leads to the question of MIX programs, which make you work a bit harder. The word "MIX" should always be handled by typing the control sequence `\MIX`. This will set it in typewriter type, namely the fixed-width font used also for examples in this manual. (Remember to type "`\MIX\`" when a blank space follows; it's the same problem as using the `\TEX` logo, see Chapter 3.)

When you want to typeset something else in typewriter type, use the abbreviation `\tt`; for example, `\MIX` is short for "`{\tt MIX}`". Or if typewriter type is being used in a math formula, you use the control sequence "`\.`", which comes in very handy. For example, "`\.{PO}`" in the excerpt from page 599 above yields the "PO" of the formula "`rII ≡ PO`". Another thing to keep in mind when doing formulas related to MIX is the fact that "rA", "rX", "rAX", "rI", and "rJ" are supposed to be in roman type, not italics; so you use the control sequences `\rA`, `\rX`, `\rAX`, `\rI`, `\rJ`. (The example above shows a typical use of "`\rI`".)

• For MIX programs themselves, further control sequences come into play. For example, let's continue with the example from page 599 of Volume 1:

```
{\yyskip\tabskip 25pt \mixthree{\!
D1@LD1@PO@\understep{D1.}\cr
@LD2@0,1(SIZE)\cr
@ENN@0,2@$.N+\.{SIZE(PO)}. $\cr
@INC@0,1@$.{P1}+\.{PO}+\.N$\cr
@LD5@0,2(TSIZE)\cr
@J5N@D4@To D4 if $.{TAG(P1)}=\hjust{`-$-$`}$.\cr
\@D2@LD5@-1,1(TSIZE)@\understep{D2.}\cr
```

and so on, ending (on page 600) with

```
@ST@-1,2(TSIZE)@$$.{SIZE(P1-1)}+\.N$,
$.{TAG(P1-1)}+\hjust{`-$-$`}$.\quad\blackslug\cr}}
```

Explanation: (i) "`\tabskip 25pt`" causes each line of the program to be indented 25 points. [For short programs, you can start with "`$$\vjust{\mixthree{\!}`" and end with "`\cr}}$$`", if you want the program to be centered. But that would be a bad idea on such a long program, because it would disallow breaks between pages.]

(ii) "`\mixthree{\!`" is the way you begin MIX program format that has three columns of special code before the right-hand column; the right-hand column is typeset normally. Sometimes there are four special columns, as in the program on page 568; in this case the first column contains numbers in italics. The rule is to use `\mixfour` when there are four such columns. The first line on page 568, for example, would be typed

```
@@SH@CON@Zero constant for initialization\cr
```

provided that you are looking at the second edition of Volume 1—the first edition has a different line there, namely

```
65@@JMP@1F@\quad$.{\RLINK(U)}=\Lambda$. \cr .
```

Sometimes, in fact, there are five special columns, as in the program on page 601; the fifth column contains centered math formulas, and for this you use `\mixfive`. (Incidentally, when a program turns out to be too wide for the normal page size, as this one does, it is typeset separately and reduced by the publisher's cameras.) At the other extreme, there sometimes are MIX programs with only two special columns; for example, to get the programs displayed at the bottom of page 242, you type

```
$$\vcenter{\mixtwo{LD1@I\cr LDA@L$+0$,1\cr}}
\qqquad\hjust{to, e.g.,}\qqquad
\vcenter{\mixtwo{LD1@I\cr LDA@BASE(0:2)\cr
STA@*+1(0:2)\cr LDA@*,1\cr}}\eqno(8)$$ .
```

(iii) When you type “\” at the beginning of a line of a MIX program, using either `\mixtwo` or `\mixthree` or `\mixfour` or `\mixfive`, it signifies a desirable place to break the page if T<sub>E</sub>X needs to make a break. The author will tell you where to put these.

(iv) “\understep” will underline a step description. This works nicely when the step doesn't involve any letters or symbols that go below the line; but otherwise you need to break the underline by brute force, discontinuing it so that it doesn't touch letters with descenders. For example, here is the proper way to type line 22 of the program on page 601 of Volume 1 (using `\mixfive` format):

```
22@R3@J3Z@DONE@1@\understep{R3. S}{\s1 p\hskip-3pt}\!
\understep{\hskip3pt 1it r@hskip 2.5pt}{\s1\hskip-2.5pt q}\!
\understep{uired?}\cr
```

The `\hskipping` brings the underlines partway under the *p* and *q*, making it look as if we have a special font with underlined symbols. This is messy in the manuscript, but it looks nice in the output; you get

```
22 R3 J3Z DONE 1 R3. Split required?
```

(See also the examples in Chapter 18 of this manual—the boldface subheads were made with such underlining. The underlines actually drawn on page 601 of the second edition of Volume 1 are too low; the third edition—typeset by T<sub>E</sub>X—will look much better!)

To sum up the last few paragraphs, we can say that MIX programs are indeed troublesome to typeset; but by using the control sequences `\mixtwo`, ..., `\mixfive`

you can avoid almost all of the difficulty of changing in and out of typewriter type and lining up the columns. Incidentally, there is also another control sequence `\mixans` that you can use for answers like number 2 on pages 523 and 524. Instead of beginning that answer with "`\ansno 2.`", you type

```
\mixans 2. { $\textcircled{S}$ HIFT $\textcircled{J}$ 5N $\textcircled{A}$ DDRERROR\cr
 $\textcircled{D}$ DEC3 $\textcircled{5}$ \cr
:
 $\textcircled{1}$ H $\textcircled{S}$ RC $\textcircled{1}$ \quad\blackslug\cr}
```

This works something like `\mixthree`, but each line begins with an additional  $\textcircled{\bullet}$ .

- For quotations you type

```
\quoteformat{<first line>\cr
<second line>\cr
:
<last line>\cr
\author{<author information>}
```

For example, the quotation at the end of Chapter 2 (Volume 1, page 463) should be done this way (including a few small changes that will be made in the third edition):

```
\quoteformat{You will, I am sure, agree with me...that if page\cr
534 finds us only in the second chapter,\cr
the length of the first one must have been really intolerable.\cr}
\author{SHERLOCK HOLMES, in {\sl The Valley of Fear} (1888)}
```

Sans-serif 8-point fonts will automatically be used for quotations typed in this way. The quotation itself is automatically set in a slanted font, while the author information is automatically set in "roman"; you can vary these conventions if necessary by typing "`\sl`" or "`\rm`".

- To insert an illustration at the top of the next convenient page, type

```
\topinsert{\vskip <height of the illustration plus a little white space>
\ctrline{\caption Fig.\ <number>. <text of the caption>}}
```

assuming that the caption fits on one line. This insertion usually goes into the manuscript just after the paragraph that first refers to this particular illustration. For example (Volume 1, page 121),

```
\topinsert{\vskip 5in
\ctrline{\caption Fig.\ 13. The \MIX\ computer.}}
```

• The following example (see Chapter 4 of this manual) shows how footnotes are treated:


```
... will never\footnote*{Well$\ldots$, hardly ever.} use the ...
```

• To get the heading "Table 1" centered on a line, type

```
\tablehead{Table 1} .
```

For the table itself, it's best to let the author tell you exactly what he wants, since there are so many possibilities. The control sequence `\9` gives a blank space equal to the width of a digit in the current roman font; this is occasionally useful when tables are being prepared.

• When you really get into typing the books, some things will occasionally arise that aren't covered here, but this might add a little spice to the task. The manuscript for Volume 2 of *ACP* (Second Edition) may be consulted for numerous examples of the recommended format.

 Here now are the TeX language definitions that explain the meanings of all these new control sequences very precisely. All of the standard definitions at the beginning of Appendix B are used, up until the font specifications, and it is unnecessary to repeat them here. The remaining definitions are:

```
\def\mod{\<\, \mathbin{\char'155\char'157\char'144}\penalty900\<\,}
\def\modulo#1{\penalty0;
  (\char'155\char'157\char'144\char'165\char'154\char'157\, \, #1)}
\def\deg{\up{\hjust{\hskip-1pt\:\w\char5}}}
```

```
\: @+cmathx \: a+cmr10 \: b+cmr9 \: c+cmr8
\: d+cmr7 \: e+cmr6 \: f+cmr5 \: g+cmi10
\: h+cmi9 \: i+cmi8 \: j+cmi7 \: k+cmi6
\: l+cmi5 \: m+cmec10 \: n+cms10 \: o+cms9
\: p+cms8 \: q+cmb10 \: r+cmb9 \: s+cmb8
\: t+cmtt \: u+cmsy10 \: v+cmsy9 \: w+cmsy8
\: x+cmsy7 \: y+cmsy6 \: z+cmsy5 \: ;+cmtit1
\: <+cms8b \: =+cms12 \: >+cms8 \: ?+cms8b
```

```
\hsize29pc \vsize45pc \maxdepth2pt \parindent19pt
\topbaseline10pt \parskip0pt plus1pt \lineskip1pt
\topskip24pt plus6pt minus10pt \botskip3pt plus6pt
```

```

\def\tenpoint{\baselineskip12pt
\disp skip12pt plus3pt minus9pt
\disp skip0pt plus3pt \disp bskip7pt plus3pt minus4pt
\def\rm{\:a} \def\sl{\:n} \def\bf{\:q} \def\it{\:g}
\def\biglp{\mathopen{\vcenter{\hjust{\:\@char'0}}}}
\def\biggrp{\mathclose{\vcenter{\hjust{\:\@char'1}}}}
\def\9{\hskip 5pt}
\mathrm adf \mathit gjl \mathsy uxz \rm}
\def\ninepoint{\baselineskip11pt
\disp skip11pt plus3pt minus8pt
\disp skip0pt plus3pt \disp bskip6pt plus3pt minus3pt
\def\rm{\:b} \def\sl{\:o} \def\bf{\:r} \def\it{\:h}
\def\biglp{\mathopen{\hjust{\:a{}}}}
\def\biggrp{\mathclose{\hjust{\:a{}}}}
\def\9{\hskip4.625pt}
\mathrm bef \mathit hkl \mathsy vyz \rm}
\def\eightpoint{\baselineskip9pt
\disp skip9pt plus3pt minus7pt
\disp skip0pt plus3pt \disp bskip5pt plus3pt minus2pt
\def\rm{\:c} \def\sl{\:p} \def\bf{\:s} \def\it{\:i}
\def\biglp{\mathopen{\hjust{\:a{}}}}
\def\biggrp{\mathclose{\hjust{\:a{}}}}
\def\9{\hskip 4.25pt}
\mathrm cef \mathit ikl \mathsy wyz \rm}
\mathex @ \def\tt{\:t}
\def\bigglp{\mathopen{\vcenter{\hjust{\:\@char'22}}}}
\def\biggrp{\mathclose{\vcenter{\hjust{\:\@char'23}}}}
\def\biggglp{\mathopen{\vcenter{\hjust{\:\@char'40}}}}
\def\biggggrp{\mathclose{\vcenter{\hjust{\:\@char'41}}}}
\def\xskip{\hskip7pt plus3pt minus4pt}
\def\yskip{\penalty-50\vskip3pt plus3pt minus2pt}
\def\yyskip{\penalty-100\vskip6pt plus6pt minus4pt}
\def\sectionskip{\penalty-200\vskip24pt plus12pt minus6pt}
\def\textindent#1{\noindent
\hjust to 19pt{\hskip0pt plus1000pt minus1000pt#1 } \!}
\def\hang{\hangindent19pt}
\def\tpage{F} \def\rhead{} \def\frstx{F} \def\csec{} \def\chd{}
\def\titlepage{\gdef\tpage{T}}
\def\runninglefthead#1{\gdef\rhead{\:m#1}}

```

```

\def\acpmark#1#2{\mark
  {\ifeven0{\hjust to .45 in{\:a\count0\hfill}\rhead\hfill\:#2}
   \else{\:a\csec\hfill\:#1\hjust to .45 in{\:a\hfill\count0}}}}
\def\runningrighthead#1 \section#2{\acpmark{\chd}{#2}
  \gdef\csec{#2} \gdef\chd{#1}}
\output{\baselineskip Opt\lineskipOpt
  \vjust to 48pc{
    \if T\tpage{
      \gdef\tpage{F}
      \vskip24pt \page \vfill \ctrline{\:c\count0}}
    \else{\baselineskip12pt \null
      \hjust to size{\ifeven0{\topmark}\else{\botmark}}
      \null \page \vfill}}
  \advcount0}

\def\sectionbegin#1{\hjust{\:<#1}\penalty1000\vskip6pt plus3pt
  \acpmark{\chd}{\csec}\noindent\tenpoint\!}
\def\dimsectionbegin#1{\yyskip
  \acpmark{\chd}{\csec}\noindent{\bf#1.}\tenpoint\xskip\!}
\def\subsectionbegin#1{\yyskip\noindent{\bf#1.}\tenpoint\xskip\!}
\def\algbegin#1(#2). {\yyskip\noindent
  {\bf #1}({\sl#2\!}){\bf.}\xskip}
\def\algstep #1. [#2]{\par\yyskip
  \hang\textindent{\bf#1.}[#2]\xskip\!}
\def\thbegin#1. {\yyskip\noindent{\bf#1.}\xskip}
\def\proofbegin{\penalty25\vskip6pt plus12pt minus4pt
  \noindent{\sl Proof.}\xskip}
\def\dproofbegin{\penalty25\noindent{\sl Proof.}\xskip}
\def\exbegin#1{\sectionskip
  \hjust{\:<#1}\penalty1000\vskip8pt minus5pt
  \gdef\frstx{T}\ninepoint}
\def\ansbegin#1{\runningrighthead{ANSWERS TO EXERCISES}
  \section{#1}\sectionskip
  \hjust{\:<SECTION #1}\penalty1000\vskip8pt minus5pt
  \acpmark{\chd}{\csec}\gdef\frstx{T}\ninepoint}
\def\anskip{\par\if T\frstx{\gdef\frstx{F}}\else{\penalty-200}
  \vskip3pt plus3pt minus1pt}
\def\oxno #1. [#2]{\anskip\textindent{\bf#1.}[{\it#2\!}]\hskip6pt}
\def\trexno #1. [#2]{\anskip\noindent\hjust to 19pt
  {\hskip-3.5pt\:\@char'170\hfill\bf#1. }[{\it#2\!}]\hskip6pt}

```



```

\def\ansno #1. {\anskip\textindent{\bf#1.}}
\def\ansnos #1,#2. {\anskip\textindent{\bf#1,}\hjust{\bf\!#2. }}
\def\quoteformat#1{\baselineskip11pt \def\rm{\:}> \def\sl{\:?}
  \vskip6pt plus2pt minus2pt {\sl\halign{\rjustline{##}\cr#1}}
\def\author#1{\penalty1000\vskip8pt plus2pt minus2pt
  \rm\rjustline{---#1}\vskip8pt plus4pt minus2pt}
\def\tablehead#1{\ctrlino{\:<#1}\ninepoint}
\def\caption Fig.\ #1. {\ninepoint{\bf Fig.\ #1.}\xskip\!}
\def\footnote#1#2{#1\botinsert{\hrule width5pc \vskip3pt
  \baselineskip9pt\hjust to size{\eightpoint#1#2\hf11}}}}
\def\star{\hjust to 0pt{\hskip 0pt minus 100pt *}}
\def\blackslug{\hjust{\hskip1pt
  \vrule width4pt height6pt depth1.5pt \hskip1pt}}

\def\MIX{\:t MIX}
\def\.\{\hjust{\:t#1}}
\def\rA{\hjust{\rm rA}} \def\rX{\hjust{\rm rX}}
\def\rAX{\hjust{\rm rAX}}
\def\rI{\hjust{\rm rI}} \def\rJ{\hjust{\rm rJ}}
\def\understep#1{\underline{\hjust{\sl#1}}}$}

\def\mixtwo#1{\ninepoint\def\\\{\noalign{\penalty-200}}
  \halign{\lft{\:t##}\quad\tabskip0pt
    @\lft{\:t##}\quad@\lft{\rm##}\cr#1}}
\def\mixthree#1{\ninepoint\def\\\{\noalign{\penalty-200}}
  \halign{\lft{\:t##}\quad\tabskip0pt
    @\lft{\:t##}\quad@\lft{\:t##}\quad@\lft{\rm##}\cr#1}}
\def\mixfour#1{\ninepoint\def\\\{\noalign{\penalty-200}}
  \halign{\rt{\it##}\quad\tabskip0pt
    @\lft{\:t##}\quad@\lft{\:t##}\quad
    @\lft{\:t##}\quad@\lft{\rm##}\cr#1}}
\def\mixfive#1{\ninepoint\def\\\{\noalign{\penalty-200}}
  \halign{\rt{\it##}\quad\tabskip0pt
    @\lft{\:t##}\quad@\lft{\:t##}\quad
    @\lft{\:t##}\quad@\ctr{$ ##$}\quad@\lft{\rm##}\cr#1}}
\def\mixans #1. #2{\def\\\{\noalign{\penalty-200}}\anskip
  \halign{\hjust to 19pt{##}@\lft{\tt##}\quad
    @\lft{\tt##}\quad@\lft{\tt##}\quad@\lft{\rm##}\cr
    {\hf11}\bf #1. }#2}}

```

## &lt;F&gt; Font tables

1. Standard "ascii" code. The American Standard Code for Information Interchange deals with characters that print and actions that don't. The following table of 128 codes shows "control" symbols for codes '001 to '032, since many computer terminals generate these symbols when the typist holds the control key down when typing a letter. These 26 codes also have other names not shown here; for example, ↑G (control-G) is also called BEL (ring the bell). It is rarely possible to transmit all 128 of these symbols from your terminal to a computer and vice versa—something strange usually happens to a few of them. But the most important ones get through.

	0	1	2	3	4	5	6	7
'000	NUL	↑A	↑B	↑C	↑D	↑E	↑F	↑G
'010	↑H	↑I	↑J	↑K	↑L	↑M	↑N	↑O
'020	↑P	↑Q	↑R	↑S	↑T	↑U	↑V	↑W
'030	↑X	↑Y	↑Z	ESC	FS	GS	RS	US
'040	SP	!	"	#	\$	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	<	=	>	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	\	]	-	-
'140	`	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	{		}	~	DEL

**2. Stanford "SUAI" code.** The following table of 128 codes (developed about 1965 at the Stanford Artificial Intelligence Laboratory) applies to numerous devices now used in the vicinity of Stanford University. For the most part it is ascii code but extended to include more printing characters. There's an unfortunate discrepancy, however, with respect to "}" ('175 in ascii, '176 at SUAI); also "~" ('176 in ascii, something like '032 at SUAI); also "-" ('136 in ascii, something like '004 at SUAI); and also "\_" ('137 in ascii, '030 at SUAI). Essentially the same code is used at Carnegie-Mellon University and at the University of Southern California, but with '176 and '175 switched. At the Massachusetts Institute of Technology the code is somewhat the same but there are ten discrepancies: codes '010, '013, '030, '032, '033, '136, '137, '175, '176, '177 are respectively called BS, ↑, ←, ≠, ESC, ^, \_, }, ~, DEL.

	0	1	2	3	4	5	6	7
'000	NUL	↓	α	β	∧	¬	ε	π
'010	λ	TAB	LF	VT	FF	CR	∞	∂
'020	⊂	⊃	∩	∪	∩	∃	⊗	↔
'030	-	→	~	≠	≤	≥	≡	∇
'040	SP	!	"	#	\$	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	<	=	>	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	\	]	↑	←
'140	'	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	{		ALT	}	BS

3. **TeX standard roman fonts.** The following table of 128 codes shows the form TeX expects its fonts to have when you use the control sequences for accents and special foreign letters listed in Chapter 9, or when you use the control sequences for upper case Greek letters listed later in this appendix. (Actually TeX never addresses codes '042, '134, '136, '137, and '173 to '177 directly; they are accessed indirectly via ligature information stored within the font itself.) Codes '043 and '044 are undefined; special characters needed in particular jobs (e.g., inverted "?" and "!" for Spanish text) might be placed there. Note that there is agreement with ascii code on all of its printing characters except for #, \$, @, \, ^, \_, {, |, }, and ~, which TeX gets from its "symbol" fonts. The same codes are used for slanted roman fonts like cms10.

	0	1	2	3	4	5	6	7
'000	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ
'010	Φ	Ψ	Ω	ι	ϰ	·	·	·
'020	·	·	·	·	·	·	·	·
'030	·	·	·	β	æ	œ	Æ	Œ
'040	ø	!	"	undefined	undefined	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	<	=	>	?
'100	Ø	Λ	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	"	]	-	—
'140	·	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	ff	fi	fl	ffi	ffl

4. **T<sub>E</sub>X typewriter fonts.** Fixed-width fonts such as the cmtt font shown below are sort of a cross between T<sub>E</sub>X roman and SUAI codes. All the accents and special characters of a T<sub>E</sub>X roman font are present except for \b, \l, \o, \ss, \H, and \O; and every ascii printing character is present. (SUAI code instead of ascii code is, however, used for the character ")", and T<sub>E</sub>X roman code is used for "~".) All SUAI characters that appear in these fonts appear in their SUAI positions, except for ω, ~, ≤, ≥, and ↓. (Furthermore you may prefer to use codes '015 and '016 in place of '140 and '047, as done in the examples of this manual.)

	0	1	2	3	4	5	6	7
'000	Γ	Δ	Θ	Λ	Β	Π	Σ	Υ
'010	Φ	Ψ	Ω	ι	ϰ	·	·	·
'020	·	·	·	·	·	ω	⊕	·
'030	·	→	≤	≥	ω	ω	£	€
'040	□	!	"	#	\$	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	<	=	>	?
'100	⊙	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	\	]	τ	←
'140	·	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	{		↓	}	'

5.  $\text{\TeX}$  standard italic fonts. The following table of 128 codes shows the form  $\text{\TeX}$  expects the italic (*it*) fonts to have in math mode, if you use the control sequences for lower case Greek letters, upper case italic Greek letters, and a few other special symbols listed later in this appendix. The same codes apply to text italic fonts like *cmtil0* and *cmu10* (the "unslanted" italic font used in the running heads of this manual). Note that there is agreement with *ascii* code on all of its printing characters, except for the % sign and the ten symbols that are missing in  $\text{\TeX}$  roman fonts (see the previous page).

	0	1	2	3	4	5	6	7
'000	$\Gamma$	$\Delta$	$\Theta$	$\Lambda$	$\Xi$	$\Pi$	$\Sigma$	$\Upsilon$
'010	$\Phi$	$\Psi$	$\Omega$	$\alpha$	$\beta$	$\gamma$	$\delta$	$\epsilon$
'020	$\zeta$	$\eta$	$\theta$	$\iota$	$\kappa$	$\lambda$	$\mu$	$\nu$
'030	$\xi$	$\pi$	$\rho$	$\sigma$	$\tau$	$\upsilon$	$\phi$	$\chi$
'040	$\imath$	$!$	$"$	$\ell$	$\wp$	$\partial$	$\mathcal{E}$	$'$
'050	$($	$)$	$*$	$+$	$,$	$-$	$.$	$/$
'060	$0$	$1$	$2$	$3$	$4$	$5$	$6$	$7$
'070	$8$	$9$	$:$	$;$	$<$	$=$	$>$	$?$
'100	$j$	$A$	$B$	$C$	$D$	$E$	$F$	$G$
'110	$H$	$I$	$J$	$K$	$L$	$M$	$N$	$O$
'120	$P$	$Q$	$R$	$S$	$T$	$U$	$V$	$W$
'130	$X$	$Y$	$Z$	$[$	$"$	$]$	$-$	$—$
'140	$'$	$a$	$b$	$c$	$d$	$e$	$f$	$g$
'150	$h$	$i$	$j$	$k$	$l$	$m$	$n$	$o$
'160	$p$	$q$	$r$	$s$	$t$	$u$	$v$	$w$
'170	$x$	$y$	$z$	$\psi$	$\omega$	$\varphi$	$\vartheta$	$\varpi$

6. **T<sub>E</sub>X standard symbol fonts.** The following table of 128 codes shows the form T<sub>E</sub>X expects the symbol (**sy**) fonts to have in math mode, if you use the control sequences for various special symbols listed later in this appendix, or if you use special keys on your terminal in math mode as explained below in subsection 8. Several positions are undefined; they can be filled with any special characters that might be needed in a particular job.

	0	1	2	3	4	5	6	7
'000	—	·	×	*	\	◦	±	≠
'010	⊕	⊖	⊗	⊘	⊙	÷	↑	•
'020	⊥	≡	⊆	⊇	∩	∪	∩	∪
'030	~	≈	∪	∩	≠	∥	∧	∨
'040	↑	→	↑	↓	↔	↔	↔	↔
'050	⇐	⇒	⇐	⇓	⇔	↗	↘	↔
'060	/	∞	∈	∉	∅	-	∠	undefined
'070	∇	∃	¬	ℵ	ℵ	∂	∂	undefined
'100	/	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ
'110	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ
'120	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ	ℵ
'130	ℵ	ℵ	ℵ	∪	∩	∩	∧	∨
'140	⊥	⊥	⊥	⊥	⊥	⊥	{	}
'150	<	>			[	]	undefined	undefined
'160	✓	‡	∇	∫	∫	∫	∫	undefined
'170	§	†	‡	¶	©	©	£	\$

7.  $\TeX$  standard extension fonts. The table of 128 codes on the next page shows the form  $\TeX$  expects the extension (ex) font to have in math mode, if you use variable delimiters or the control sequences for large operators listed later in this appendix.

Actually  $\TeX$  addresses most of these characters indirectly; for example, all left parentheses are addressed starting with character '000, based on information stored in the font itself, and the font also tells  $\TeX$  that arbitrarily large left parentheses can be made from characters '060 (top), '102 (middle), '100 (bottom). The only codes explicitly referred to by  $\TeX$  are '000 to '016, '110, '112, '114, '116, '120 to '127, and '160. Thus, a font designer can move most of the other symbols if desired, subject only to the restriction that the code number of a large symbol be greater than the code numbers of its smaller variants. (If codes are changed, however, it may be necessary to change the definitions of control sequences like `\biggl p` in Appendix B.) It is expected that positions marked "undefined" in this chart will be filled with characters specially tailored to specific jobs; for example, character '177 is used for the "dangerous bend" symbol in this manual, but it might not be present in all  $\TeX$  extension fonts.



	0	1	2	3	4	5	6	7
'000	(	)	[	]	L	J	Γ	Γ
'010	{	}	<	>			/	undefined
'020	(	)	(	)	[	]	L	J
'030	Γ	Γ	{	}	<	>	/	undefined
'040	(	)	[	]	L	J	Γ	Γ
'050	{	}	<	>	/	undefined	undefined	undefined
'060	f	\	Γ	Γ	L	J		
'070	f	\	l	j	{	}	·	undefined
'100	\	)			undefined	undefined	U	U
'110	§	§	⊙	⊙	⊕	⊕	⊗	⊗
'120	Σ	Π	∫	U	∩	⊕	∧	∨
'130	Σ	Π	∫	U	∩	⊕	∧	∨
'140	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined
'150	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined
'160	✓	✓	✓	✓	∨		Γ	undefined
'170	▶	↯	˘	˘	˘	˘	undefined	undefined

8. **T<sub>E</sub>X math mode.** When T<sub>E</sub>X is in math mode, it converts 7-bit codes into 9-bit codes according to the table below. Furthermore a "type" is associated with the 9-bit code, making (almost) a 12-bit code, since there are seven types (Ord, Op, Bin, Rel, Open, Close, Punct); see Chapter 18. This conversion is based on the SUAI code. For example, if you type "→" at Stanford (character '031) the table below says that this is converted to Rel 441, namely a mathematical "relation" found in the sy font as character '041, and this is "→".

Not all of these 128 codes can appear in character files that are prepared with ordinary software, but the chart shows what would happen if they could. If people at MIT ever want to use T<sub>E</sub>X they will undoubtedly make changes to the internal table that T<sub>E</sub>X uses for this conversion, because of the ten discrepancies between MIT's code and the SUAI code. However, people at CMU or USC should have no trouble, since T<sub>E</sub>X uses this table only for characters classified as (letter) or (otherchar) (see Chapter 9).

	0	1	2	3	4	5	6	7
'000	Bin 401	Rel 449	Ord 219	Ord 214	Bin 596	Ord 472	Ord 217	Ord 291
'010	Ord 225	Ord 215	Ord 216	Op 569	Bin 406	Bin 410	Ord 461	Ord 245
'020	Rel 492	Rel 499	Bin 534	Bin 599	Ord 470	Ord 471	Bin 412	Rel 444
'030	Ord 465	Rel 441	Rel 490	Rel 494	Rel 424	Rel 425	Rel 421	Bin 597
'040	Ord 469	Close 041	Ord 541	Ord 561	Ord 577	Ord 045	Ord 046	Close 047
'050	Open 050	Close 051	Ord 052	Bin 059	Punct 054	Bin 400	Ord 056	Ord 057
'060	Ord 060	Ord 061	Ord 062	Ord 069	Ord 064	Ord 065	Ord 066	Ord 067
'070	Ord 070	Ord 071	Ord 072	Punct 079	Rel 074	Rel 075	Rel 076	Close 077
'100	Ord 974	Ord 901	Ord 902	Ord 909	Ord 904	Ord 905	Ord 906	Ord 907
'110	Ord 910	Ord 911	Ord 912	Ord 919	Ord 914	Ord 915	Ord 916	Ord 917
'120	Ord 920	Ord 921	Ord 922	Ord 929	Ord 924	Ord 925	Ord 926	Ord 927
'130	Ord 930	Ord 931	Ord 932	Open 199	Bin 404	Close 195	Rel 442	Rel 440
'140	Open 140	Ord 941	Ord 942	Ord 949	Ord 944	Ord 945	Ord 946	Ord 947
'150	Ord 950	Ord 951	Ord 952	Ord 959	Ord 954	Ord 955	Ord 956	Ord 957
'160	Ord 960	Ord 961	Ord 962	Ord 969	Ord 964	Ord 965	Ord 966	Ord 967
'170	Ord 970	Ord 971	Ord 972	Open 546	Ord 552	Bin 405	Close 547	Bin 017

**9. Control sequences.** The tables we have seen show all of the special symbols that appear in TeX's standard fonts. But the question remains, how can a person specify them on an ordinary keyboard? Well, you can always define your favorite control sequence in terms of the `\char` operation; and if you have a suitable keyboard you can type the symbols of SUAI code directly. TeX also recognizes the control sequences listed below, when in math mode.

(a) *Lower case Greek letters:*

$\alpha$	<code>\alpha</code>	$\kappa$	<code>\kappa</code>	$\upsilon$	<code>\upsilon</code>
$\beta$	<code>\beta</code>	$\lambda$	<code>\lambda</code>	$\phi$	<code>\phi</code>
$\gamma$	<code>\gamma</code>	$\mu$	<code>\mu</code>	$\chi$	<code>\chi</code>
$\delta$	<code>\delta</code>	$\nu$	<code>\nu</code>	$\psi$	<code>\psi</code>
$\epsilon$	<code>\epsilon</code>	$\xi$	<code>\xi</code>	$\omega$	<code>\omega</code>
$\zeta$	<code>\zeta</code>	$\pi$	<code>\pi</code>	$\varphi$	<code>\varphi</code>
$\eta$	<code>\eta</code>	$\rho$	<code>\rho</code>	$\vartheta$	<code>\vartheta</code>
$\theta$	<code>\theta</code>	$\sigma$	<code>\sigma</code>	$\varpi$	<code>\varpi</code>
$\iota$	<code>\iota</code>	$\tau$	<code>\tau</code>		

(b) *Upper case Greek letters:*

$\Gamma$	<code>\Gamma</code>	$\Sigma$	<code>\Sigma</code>	$\Gamma$	<code>\Gamma</code>	$\Sigma$	<code>\Sigma</code>
$\Delta$	<code>\Delta</code>	$\Upsilon$	<code>\Upsilon</code>	$\Delta$	<code>\Delta</code>	$\Upsilon$	<code>\Upsilon</code>
$\Theta$	<code>\Theta</code>	$\Phi$	<code>\Phi</code>	$\Theta$	<code>\Theta</code>	$\Phi$	<code>\Phi</code>
$\Lambda$	<code>\Lambda</code>	$\Psi$	<code>\Psi</code>	$\Lambda$	<code>\Lambda</code>	$\Psi$	<code>\Psi</code>
$\Xi$	<code>\Xi</code>	$\Omega$	<code>\Omega</code>	$\Xi$	<code>\Xi</code>	$\Omega$	<code>\Omega</code>
$\Pi$	<code>\Pi</code>			$\Pi$	<code>\Pi</code>		

(c) *Script letters:*

$\mathcal{A}$	<code>\mathcal{A}</code>	$\mathcal{J}$	<code>\mathcal{J}</code>	$\mathcal{Y}$	<code>\mathcal{Y}</code>
$\mathcal{B}$	<code>\mathcal{B}</code>	$\mathcal{K}$	<code>\mathcal{K}</code>	$\mathcal{T}$	<code>\mathcal{T}</code>
$\mathcal{C}$	<code>\mathcal{C}</code>	$\mathcal{L}$	<code>\mathcal{L}</code>	$\mathcal{U}$	<code>\mathcal{U}</code>
$\mathcal{D}$	<code>\mathcal{D}</code>	$\mathcal{M}$	<code>\mathcal{M}</code>	$\mathcal{V}$	<code>\mathcal{V}</code>
$\mathcal{E}$	<code>\mathcal{E}</code>	$\mathcal{N}$	<code>\mathcal{N}</code>	$\mathcal{W}$	<code>\mathcal{W}</code>
$\mathcal{F}$	<code>\mathcal{F}</code>	$\mathcal{O}$	<code>\mathcal{O}</code>	$\mathcal{X}$	<code>\mathcal{X}</code>
$\mathcal{G}$	<code>\mathcal{G}</code>	$\mathcal{P}$	<code>\mathcal{P}</code>	$\mathcal{Y}$	<code>\mathcal{Y}</code>
$\mathcal{H}$	<code>\mathcal{H}</code>	$\mathcal{Q}$	<code>\mathcal{Q}</code>	$\mathcal{Z}$	<code>\mathcal{Z}</code>
$\mathcal{I}$	<code>\mathcal{I}</code>	$\mathcal{R}$	<code>\mathcal{R}</code>	$\mathcal{e}$	<code>\mathcal{e}</code>

*(d) Binary operators:*

$\pm$	<code>\pm</code>	$\oplus$	<code>\oplus</code>	$*$	<code>\ast</code>
$\mp$	<code>\mp</code>	$\ominus$	<code>\ominus</code>	$\circ$	<code>\circ</code>
$\times$	<code>\times</code>	$\otimes$	<code>\otimes</code>	$\bullet$	<code>\bullet</code>
$\div$	<code>\div</code>	$\oslash$	<code>\oslash</code>	$\intercal$	<code>\intercal</code>
$\backslash$	<code>\backslash</code>	$\odot$	<code>\odot</code>	$\sqcup$	<code>\sqcup</code>
$\cdot$	<code>\cdot</code>	$\uplus$	<code>\uplus</code>	$\sqcap$	<code>\sqcap</code>

*(e) Binary relations:*

$\uparrow$	<code>\up</code>	$\perp$	<code>\perp</code>	$\prec$	<code>\prec</code>
$\downarrow$	<code>\down</code>	$\vdash$	<code>\vdash</code>	$\preceq$	<code>\preceq</code>
$\Leftrightarrow$	<code>\Leftrightarrow</code>	$\dashv$	<code>\dashv</code>	$\succ$	<code>\succ</code>
$\Rightarrow$	<code>\Rightarrow</code>	$\dashv$	<code>\dashv</code>	$\succeq$	<code>\succeq</code>
$\Uparrow$	<code>\Uparrow</code>	$\mapsto$	<code>\mapsto</code>	$\sqsubset$	<code>\sqsubset</code>
$\Updownarrow$	<code>\Updownarrow</code>	$\relv$	<code>\relv</code>	$\leqslant$	<code>\leqslant</code>
$\Downarrow$	<code>\Downarrow</code>	$\relvv$	<code>\relvv</code>	$\gtrless$	<code>\gtrless</code>
$\Rrightarrow$	<code>\Rrightarrow</code>	$\subset$	<code>\subset</code>	$\gtrgtr$	<code>\gtrgtr</code>
$\lsh$	<code>\lsh</code>	$\supset$	<code>\supset</code>	$\simeq$	<code>\simeq</code>
$\rsh$	<code>\rsh</code>	$\in$	<code>\in</code>	$\approx$	<code>\approx</code>
		$\notin$	<code>\notin</code>	$\doteq$	<code>\doteq</code>

You can also use the control sequence `\not` to negate or “cross out” most of the relations above. For example, the symbol “ $\not\subset$ ” is really two symbols, obtained by typing “`\not\subset`”. (Character ‘100 in the symbol font has a width of zero, so it will overlap the following character.) But watch out: you should actually type “`\mathrel{\not\subset}`”, in order to prevent TeX from breaking a line after `\not`. (See the definition of `\neq` in Appendix B.)

*(f) Brackets:*

$\lfloor$	<code>\lfloor</code>	$\rfloor$	<code>\rfloor</code>
$\lceil$	<code>\lceil</code>	$\rceil$	<code>\rceil</code>
$\{$	<code>\{</code>	$\}$	<code>\}</code>
$\langle$	<code>\langle</code>	$\rangle$	<code>\rangle</code>
$\llbracket$	<code>\llbracket</code>	$\rrbracket$	<code>\rrbracket</code>
$\lvert$	<code>\lvert</code>	$\rvert$	<code>\rvert</code>
$\lll$	<code>\lll</code>	$\rrr$	<code>\rrr</code>

(g) "Large" operators (text and display styles):

$\Sigma$	$\sum$	<code>\sum</code>	$\cap$	$\bigcap$	<code>\inter</code>	$\prod$	$\prod$	<code>\prod</code>
$\oplus$	$\bigoplus$	<code>\osum</code>	$\cup$	$\bigcup$	<code>\union</code>	$\otimes$	$\bigotimes$	<code>\oproduct</code>
$\int$	$\int$	<code>\int</code>	$\sqcup$	$\bigsqcup$	<code>\squnion</code>	$\odot$	$\bigodot$	<code>\odotprod</code>
$\oint$	$\oint$	<code>\oint</code>	$\wedge$	$\bigwedge$	<code>\meet</code>	$\uplus$	$\biguplus$	<code>\munion</code>
			$\vee$	$\bigvee$	<code>\join</code>			

(h) Miscellaneous math symbols:

$\imath$	<code>\iit</code>	$\infty$	<code>\infty</code>	$\partial$	<code>\partial</code>
$\jmath$	<code>\jit</code>	$\emptyset$	<code>\emptyset</code>	$\nabla$	<code>\nabla</code>
$\Re$	<code>\real</code>	$\#$	<code>\#</code>	$\int$	<code>\smallint</code>
$\Im$	<code>\imag</code>	$\parallel$	<code>\parallel</code>	$\surd$	<code>\surd</code>
$\aleph$	<code>\aleph</code>	$\angle$	<code>\angle</code>	$\top$	<code>\top</code>
$\wp$	<code>\wp</code>	$\prime$	<code>\prime</code>	$\perp$	<code>\bot</code>

(i) Miscellaneous nonmath symbols (but allowed only in math mode):

$\S$	<code>\section</code>	$\textcircled{\circ}$	<code>\circ</code>
$\dagger$	<code>\dag</code>	$\textcircled{C}$	<code>\copyright</code>
$\ddagger$	<code>\ddag</code>	$\text{£}$	<code>\sterling</code>
$\P$	<code>\P</code>	$\text{\$}$	<code>\\$</code>

Some of the symbols in TeX's math fonts can be accessed directly only by using the SUAI-oriented conversions in subsection 8. For example, the only way to get a left arrow is by typing "\$+"; no built-in control sequence has been defined for it. If your keyboard doesn't have this symbol, the remedy is to define an appropriate new control sequence, such as

```
\def\from{\mathrel{\char'440}} .
```

**<H> Hyphenation**

The conditions under which  $\TeX$  will try to hyphenate a word are discussed in Chapter 14. Now let's consider how hyphenation is actually accomplished.

It seems to be undesirable to look for the set of all possible places to hyphenate every given word. For one thing, the problem is extremely difficult, since the word "record" is supposed to be broken as "rec-ord" when it is a noun but "re-cord" when it is a verb. We might consider also the word "hyphenation" itself, which appears to be rather an exception:

hy-phen-a-tion vs. con-cat-e-na-tion .

Why does the "n" go with the "a" in one case and not the other? Starting at letter **a** in the dictionary and trying to find rigorous rules for hyphenation without much knowledge, we come up against **a-part** vs. **ap-er-ture**, **aph-o-rism** vs. **a-pha-sia**, etc. It becomes clear that what we want is not an accurate but ponderously slow routine that consumes a lot of memory space and processing time; instead we want a set of hyphenation rules that are

- a) simple enough to explain in a couple of pages,
- b) almost always safe,
- and c) powerful enough that bad breaks due to missed hyphenations are very rare.

Point (c) means that a proofreader's job should be only negligibly more difficult than it would be if an intelligent human being were doing all of the hyphenations needed to typeset the same material.

So here are the rules  $\TeX$  uses (found with the help of Frank Liang):

1) *Exception removal.* If the first seven letters of the word appear in a small internal dictionary of words to be treated specially (about 350 words in all, see below), use the hyphenation found in that dictionary. Furthermore some of the entries in the dictionary specify looking at more than seven letters to make sure that the exception is real; e.g., "in-form-ant" wouldn't be distinguished from the unexceptional word "in-for-ma-tion" on the basis of seven letters alone. If the given word has seven letters or fewer and ends with "s", the word minus the s is also looked up. The dictionary contains nearly all the common English words for which the rules below would make an incorrect break, plus additional words that are common in computer science writing and whose breaks are not satisfactorily found by the rules.

2) *Suffix removal.* A permissible hyphen is inserted if the word ends with **-able** (preceded by e, h, i, k, l, o, u, v, w, x, y or nt or rt), **-ary** (preceded by ion

or en), -cal, -cate (preceded by a vowel), -cial, -cious (unless preceded by s), -cient, -dent, -ful, -ize (preceded by l), -late (preceded by a vowel), -less, -ly, -ment, -ness, -nary (unless preceded by e or io), -ogy, -rapher, -raphy, -scious, -scope, -scopic, -sion, -sphere, -tal, -tial, -tion, -tion-al, -tive, -ture. Here a "vowel" is either a, e, i, o, u, or y; the other 20 letters are "consonants."

There is also a somewhat more complex rule for words ending with "ing": If ing is preceded by fewer than four letters, insert no permissible hyphens. Otherwise if ing is preceded by two identical consonants other than f, l, s, or z, break between them. Otherwise if it is preceded by a letter other than l, break the -ing. Otherwise if the letter before ling is b, c, d, f, g, k, p, t, or z, break before this letter (except break ck-ling if the word ends with ckling). Otherwise break -ing.

Furthermore the same suffix removal routine is applied to the residual word after having successfully found the suffixes -able, -ary, -ful, -ize, -less, -ly, -ment, and -ness. If the original word ends in s and no suffix was found, the final s is removed and the suffix routine is applied again. If the original word ends in ed the suffix routine is applied to the word with the final d removed, and (if that is unsuccessful) to the word with final ed removed.

Any suffixes found are effectively removed from the word, not examined by rules 3 and 4. If the original word ends with e or s or ed, this final letter or pair of letters is also effectively removed.

3) *Prefix removal.* A permissible hyphen is inserted if the word begins with be- (followed by c, h, s, or w), com-, con-, dis- (unless followed by h or y), equi- (unless followed by v), equiv-, ex-, hand-, horse-, hyper-, im-, in- (but use in\*ter- or in\*tro- if present), lex\*i-, mac\*ro-, math\*e-, max\*i-, min\*i-, mul\*ti-, non-, out-, over-, pseu\*do-, quad-, semi-, some-, sub-, su\*per-, there-, trans- (followed by a, f, g, l, or m), tri- (followed by a, f, or u), un\*der-, un- (unless followed by der or i). Here an asterisk denotes a second permissible hyphen to be recognized, but only if the entire prefix appears.

After the prefixes dis-, im-, in-, non-, over-, un- have been recognized as stated, the prefix routine is entered again. Any prefixes found are effectively removed from the word, and not examined by rule 4.

4) *Study of consonant pairs.* In the remainder of the word, after suffixes and prefixes have been removed, we combine the letter pairs ch, gh, ph, sh, th, treating them as single consonants.

If the three-letter combination XYY is found, where X is a vowel and Y a consonant, break between the Y's, except if Y is l or s. In the latter case, break only if the following letter is a vowel and the word doesn't end "XYYer" or "XYYers".

If the three-letter combination Xck is found, where X is a vowel, break after the ck.

If the three-letter combination Xqu is found, where X is a vowel, break before the qu.

If the four-letter combination XYZW is found, where X and W are vowels and Y and Z are consonants, break between the consonants unless YZ is one of the following pairs:

bl, br, cl, cr, chl, chr, dg, dr, fl, fr, ght, gl, gr, kn, lk, lq, nch, nk, nx, phr, pl, pr, rk, sp, sq, tch, tr, thr, wh, wl, wn, wr.

Furthermore do not break between the consonants if the word ends with XYZer, XYZers, XYZage, XYZages, or XYZest, when YZ is one of the pairs

ft, ld, mp, nd, ng, ns, nt, rg, rm, rn, rt, st.

5) *Retaining short ends.* After applying rules 1, 2, 3, and 4, take back all "permissible" breaks that result in only one or two letters after the break, or that have only one letter before it, or that have only one letter between prefix and suffix. (Thus, for example, the suffix rule will break -ly, but this won't count in the final analysis; it does affect the hyphenation algorithm, however, since the suffixes in words like "rationally" will be found by repeated suffix removal.)

Also, take back any break leading to the syllable -e, -Xe, or -XYe, where X and Y are any two letters and where this e occurs at the end of the shortest subword on which suffix removal was tried in rule 2. (This rule avoids syllables with "silent e". For example, we do not wish to hyphenate rid-dle, proces-ses, was-teful, arran-gement, themsel-ves, lar-gely, and so on.) Similarly, final syllables of the form -Xed or -XYed (except -ized) are also disregarded.

#### Example of hyphenation:

su-per-califragilis-ticex-pialido-cious.

(This is a correct subset of the "official" syllabification specified by the coiners of this word, namely su-per-cal-i-frag-il-is-tic-ex-pi-al-i-do-cious.)

Now here's the dictionary of words that should be handled separately, as mentioned in rule 1. (When an asterisk appears, it means that this letter is checked too, in addition to the first seven letters.)

First, we include the following words since they are exceptions to the suffix rules:

(-able) con-trol-lable eq-uable in-sa-tiable ne-go-tiable  
so-ciable turn-table un-con-trollable un-so-ciable  
(-dent) de-pend-ent in-de-pend-ent



- (-ing) any-thing bal-ding dar-ling dump-ling err-ing eve-ning  
 every-thing far-thing found-ling ink-ling main-<sup>7</sup>spring  
 nest-ling off-spring play-thing sap-ling shoe-string  
 sib-ling some-thing star-ling ster-ling un-err-ing  
 up-swing weak-ling year-ling
- (-ize) civ-i-lize crys-tal-ize im-mo-bi-lize me-ta-bo-lize  
 mo-bi-lize mo-nop-o-lize sta-bi-li-ze tan-ta-lize  
 un-civ-i-lized
- (-late) pal-ate
- (-ment) in-clem-ent
- (-ness) bar-on-ess li-on-ess
- (-ogy) eu-logy ped-a-gogy
- (-scious) lus-cious
- (-sphere) at-mos-phere
- (-tal) met-al non-metal pet-al post-al rent-al
- (-tion) cat-ion
- (-tive) com-bat-ive
- (-ture) stat-ure

Exceptions to the prefix rules:

- (be-) beck-on bes-tial
- (com-) com-a-tose come-back co-me-dian comp-troller
- (con-) cone-flower co-nun-drum
- (equi-) equipped
- (hand-) handle-bar
- (in-) inch-worm ink-blot inn-keeper
- (inter-) in-te-rior
- (mini-) min-is-ter min-is-try
- (non-) none-the-less
- (quad-) qua-drille
- (some-) som-er-sault
- (super-) su-pe-rior
- (un-) u-na-nim-ity u-nan-i-mous unc-tuous

Exceptions to the consonant rules:

- bt: debt-or
- ck: ac-know-ledge

ct: de-duct-i\*ble ex-act-i-tude in-ex-act-i-tude  
 pre-dict-\*able re-spect-\*able un-pre-dict-able vict-ual  
 dl: needle-work idler  
 ff: buff-er off-beat off-hand off-print off-shoot off-shore stiff-en  
 ft: left-ist left-over lift-off  
 fth: soft-hearted  
 gg: egg-nog egg-head  
 gn: cognac for-eign-er vignette  
 gsh: hogs-head  
 ld: child-ish old-est hold-out hold-over hold-up  
 lf: self-ish  
 ll: bull-ish crest-fallen dis-till-\*ery fall-out lull-aby  
 roll-away sell-out wall-eye  
 lm: psalm-ist  
 ls: else-where false-hood  
 lt: con-sult-ant volt-age  
 lv: re-solv-able re-volv-er solv-able un-solv-able  
 mb: beach-comber bomb-er climb-er plumb-er  
 mp: damp-en damp-est  
 nch: clinch-er launch-er lunch-eon ranch-er trench-ant  
 nc: an-nouncer bouncer fencer hence-forth mince-meat si-lencer  
 nd: bind-ery bound-ary com-mend-\*a-\*t\*ory de-pend-able  
 ex-pend-able fiend-ish land-owner out-land-ish round-about  
 send-off stand-out  
 ng: change-over hang-out hang-over ha-rangue me-ringue  
 orange-ade tongue venge-ance  
 ns: sense-less  
 nt: ac-count-ant ant-acid ant-eater count-ess per-cent-\*age  
 rep-re-sentative  
 nth: ant-hill pent-house  
 pt: ac-cept-able ac-ceptor adapt-able adapt-er crypt-analysis  
 in-ter-ru\*p\*t-\*i\*ble  
 qu: an-tiq-uity ineq-uity iniq-uity liq-uefy liq-uid liq-ui-date  
 liq-ui-da-tion liq-uor pre-req-ui-site req-ui-si-tion  
 sub-sequence u-biq-ui-tous  
 rb: ab-sorb-ent carb-on herbal im-per-turb-able  
 rch: arch-ery arch-an-gel re-search-er un-search-able

rd: ac-cord-ance board-or chordal hard-en hard-est haz-ard-ous  
jeop-ard-ize re-corder stand-ard-ize stew-ard-ess yard-age

rf: surf-er

rg: morgue

rl: curl-i-cue

rm: af-firm-a\*t\*i\*ve con-form-\*ity de-form-ity in-form-a\*nt  
non-con-form-ist

rn: cav-ern-ous dis-cern-ible mod-ern-ize turn-about turn-over  
un-gov-ern-able west-ern-ize

rp: harp-ist sharpen

rq: torque

rs: coars-en ir-re-vers-ible nurse-maid nurs-ery  
re-hears-al re-vers-ible wors-en

rt: art-ist con-vert-ible court-yard fore-short-en heart-ache  
heart-ily short-en

rth: apart-heid court-house earth-en-ware north-east north-ern  
port-hole

rv: nerv-ous ob-serv-a\*ble ob-server pre-serv-\*a\*t\*i\*ve serv-er  
serv-ice-able

sch: pre-school

sc: con-de-scend cre-scendo de-cre-scendo de-scend-ent de-scent  
pleb-i-scite re-scind sea-scape

sk: askance snake-skin whisk-er

sl: cole-slaw

sn: rattle-snake

ss: class-ify class-room cross-over dis-miss-al ex-press-ible  
im-pass-able less-en pass-able toss-up un-class-i-fied

st: ar-mi-stice astig-ma-tism astir astonish-ment blast-off  
by-stand-er candle-stick cast-away cast-off con-test-ant  
co-star de-test-able di-gest-ible east-ern ex-ist-ence  
fore-stall in-con-test-able in-di-ges\*t-\*i\*ble  
in-ex-haust-ible life-style lime-stone live-stock mile-stone  
non-ex-ist-ent per-sist-ent pho-to-stat re-start-ed  
re-state-ment re-store shy-ster side-step smoke-stack  
sug-gest-\*i\*ble thermo-stat waste-bas-ket waste-land

sth: mast-head post-hu-mous priest-hood

sw: side-swipe

tt: watt-meter

tw: be-tween  
tz: kib-itzer  
zz: buzz-er

Of course, this is not a complete list of exceptions. But it does seem to cover all words that have a reasonably high chance of being mis-hyphenated in T<sub>E</sub>X's output, considering the fact that T<sub>E</sub>X usually finds a good way to break a paragraph without any hyphenation at all.

The following words have been also been included in the special dictionary, because they are common in the author's vocabulary, and because they need more hyphens than T<sub>E</sub>X would otherwise find:

al-go-rithm	es-tab-lish	prob-abil-ity
bib-li-og-raphy	gen-er-ator	prob-able
bi-no-mial	hap-hazard	pro-ce-dure
cen-ter	neg-li-gible	pub-li-ca-tion
com-put-a-*bil-ity	pe-ri-odic	pub-lish
dec-la-ra-tion	poly-no-mial	re-place-ment
de-gree	pre-vious	when-ever

## &lt;I&gt; Index

This index includes all control sequences known to  $\text{\TeX}$  or defined in Appendix B, and it also lists error messages that are mentioned outside of Chapter 27.

- $\backslash A$  (circumflex), 38, 64, 132.
- $\backslash a$  (Scandinavian accent), 38–39, 132.
- Abbreviations, 49, 154.
- $\backslash above$  (general fraction), 68, 134, 139.
- Absolute value, 76, 84, 87.
- $\backslash accent$  (nonstandard accent character), 36, 82, 132.
- $\langle accent \rangle$ , 123, 132, 143.
- Accents, 8, 10, 35–36, 38–39, 44, 53–54, 123, 143.
  - in math formulas, 64, 132–133.
  - table, 38.
- Acute accent, 8, 10, 35, 132.
- $\backslash advcount$  (advance a counter), 60, 111, 113, 120, 129, 137.
- $\backslash AE$  ( $\mathcal{A}$ ), 37, 122.
- $\backslash ae$  ( $\mathcal{a}$ ), 37, 122, 148.
- after, 56, 120, 129.
- $\backslash aleph$  ( $\aleph$ ), 10, 179.
- Alignment, 104–109, 117–118, 126–127, 135, 140, 144.
- $alignsize$ , 144.
- $\backslash alpha$  ( $\alpha$ ), 35, 61, 132, 177.
- $\langle alt-mode \rangle$  (ALT), 32, 165.
- ! Ambiguous..., 134, 139.
- $\backslash angle$  ( $\angle$ ), 179.
- Angle brackets, 41, 75, *see also*  $\backslash angle$ ,  $\backslash rangle$ .
- Angstrom unit, 38–39.
- Answers to the exercises, 148–151.
- Apostrophe, 4.
- $\backslash approx$  ( $\approx$ ), 52, 61, 178..
- Argument, 97.
- Ascii, 168.
- $\backslash Asc$  ( $\mathcal{A}$ ), 10, 177.
- $\backslash ast$  ( $*$ ), 178.
- AT&T, 108.
- $\langle atom \rangle$ , 84, 132.
- $\backslash atop$  (ruleless fraction), 67, 71, 134, 139, 149.
- $\backslash b$  (vector accent), 38, 64, 132.
- Backslash, 7, 28.
- Backspace, 45, *see also*  $\langle delete \rangle$ , Negative glue,  $\backslash sponse$ .
- Badness, 47, 55, 58.
- Bar accent, 38–39, 64, 132.
- Baseline, 14–15, 42–45.
- $\backslash baselineskip$  (normal vertical spacing between baselines), 18, 58, 60, 100, 109, 112, 115, 118, 119, 127, 128, 136, 149, 153.
- Basic  $\text{\TeX}$  format, 10, 19, 151–153.
- basic.TEX, 19, 151–153.
- $\backslash beta$  ( $\beta$ ), 61, 132, 177.
- $\backslash bf$  (boldface), 12, 153.
- Bibliographic references, 15, 154.
- $\backslash biggglp$  (superlarge left parenthesis), 78, 153.
- $\backslash biggrpr$  (superlarge right parenthesis), 78, 153.
- $\backslash bigglp$  (large left parenthesis), 77–78, 96, 149, 153.
- $\backslash biggrp$  (large right parenthesis), 78, 96, 149, 153.
- $\backslash biglp$  (largish left parenthesis), 78, 81–82, 97, 149, 153.
- $\backslash bigrp$  (largish right parenthesis), 78, 81–82, 97, 149, 153.
- Bin box, 82–85, 132–133, 176.
- Binary operation, 55–56, 82–83, 96, 178.
- Black box, 43.
- $\backslash blacklug$  ( $\blacksquare$ ), 158, 167.
- Blank delimiter, 75–77.
- Blank line, 23, 28, 32.
- Blank space character, 5, 8–9, 12, 17, 25, 28, 30–33, 38, 41, 61, 80, 98, 106, 114.
  - summary, 33.
- $\backslash bot$  ( $\perp$ ), 179.
- $\backslash botinsert$  (insertion at bottom of page), 52, 54, 57, 59, 118, 127, 167.

- `\botmark` (current mark at bottom of page), 110–111, 117, 126, 135, 166.  
`\botskip` (glue above `\botinsert`), 59, 119, 126, 136.  
 Bound insertion, 127.  
`\box` (`\saved box`), 101, 102, 115–116, 124, 133.  
`\box`, 115, 124, 133.  
 Boxes, 41–46, 99–103.  
`\boxit`, 101.  
 Braces, variable-width horizontal, 103, 174.  
 Braces, variable-width vertical, 75, 174.  
 Breaking lists of lines into pages, 57–60.  
 Breaking math formulas, in text, 54–55, 84–85, 120, 129.  
     in displays, 94–96.  
 Breaking paragraphs into lines, 25–26, 52–57, 71, 101.  
 Breaks between pages, legal, 57.  
 Breaks in paragraph, feasible, 55.  
     legal, 54–55.  
 Breve accent, 38–39, 132.  
`\Bscr` ( $\mathfrak{B}$ ), 10, 177.  
`\bullet` ( $\bullet$ ), 159, 178.  
  
`\c` (cedilla accent), 23, 38–39, 132.  
 Calculus, 81–82.  
 Capacity of TeX, 143–144.  
`\carriage-return`, 12.  
`\carriage-return` (CR), 21, 28, 29, 32, 95, 165.  
`\cdot` ( $\cdot$ ), 178.  
`\cdots` ( $\cdots$ ), 86, 149, 152.  
`\cdots` (`\cdots` followed by space), 86–87, 152.  
 Cedilla accent, 23, 37–39.  
`\char` (specified character number), 34, 73–74, 79, 84, 116, 123, 132.  
`\char`, 116.  
 Character categories, table, 28.  
 Character conversion in math mode, 35, 60, 176.  
`\chcode` (change category code), 18, 32, 115, 119, 128, 131, 137, 141, 148, 151.  
`\chi` ( $\chi$ ), 4, 149, 177.  
`\choose` (binomial coefficient), 67–69, 149, 152.  
`\chop` (change depth of box), 78, 150, 152.  
  
`\chpar` (change TeX integer parameter), 56, 59, 85, 115, 119–120, 128–129, 137, 141.  
`\circ` ( $\circ$ ), 178.  
 Circumflex accent, 38, 64, 132.  
 Close box, 82–84, 133, 176.  
`cm` (centimeter), 10, 40–41.  
`cmathx`, 103, 153, 174–175.  
`cmb10`, 36, 153, 170.  
`cmi10`, 74, 132, 172.  
`cmr10`, 14, 26, 36, 42, 103, 153, 170.  
`cms10`, 36, 42, 153, 170.  
`cmss10`, 36, 170.  
`cmsy10`, 153, 173.  
`cmti10`, 74, 153, 172.  
`cmul0`, 172.  
 Colon, 48.  
 Colon-equal operator, 91.  
`\comb` (combinatorial formula), 64, 69, 134, 139.  
 Comma, 48, 71–72, 82, 85.  
 Computer Modern fonts, 14, 170.  
 Conditional text, 111–114, 121, 130, 137, 140, 150.  
 Consonant, 181.  
 Contents of this manual, table, 3.  
 Continued fractions, 69.  
 Control sequences, 8–12, 21, 27, 30–31, 33, 98, 114–115, 122, 131.  
     complete list, 187–197.  
     how to define, 96–99.  
     tracing, 147.  
 Control space, 8–9, 12, 33, 49, 81, 123, 133.  
 Conversion of characters in math mode, 35, 60, 176.  
`\copyright` ( $\copyright$ ), 61, 179.  
`\cos` (cosine operator), 72–73, 151.  
`\cot` (cotangent operator), 72, 151.  
`\count` (value of a counter), 34, 60, 111–113, 120, 129, 137.  
 Counters, 111, 120, 129, 137.  
`\cpile` (centered pile), 89, 93, 152.  
`\cr` (end of row or column to be aligned), 88–89, 104–107, 117–118, 126–127, 135, 140, 144.  
`\csc` (cosecant operator), 72, 151.  
`\Cscr` ( $\mathfrak{C}$ ), 177.  
`\ctr` (centerify), 88–89, 104–106, 151.

- `\ctrline` (centered line), 15-16, 20-21, 26-27, 50, 112, 151.  
 Cube root, 63.  
 Current font selections, 12-15, 18, 36, 73, 115, 118-119, 127-128, 118-119, 127-128, 140, 145.  
 Current math fonts, 73-74, 118-119, 127-128.  
  
`\dag` ( † ), 61, 179.  
 Dangerous bend, 2, 47, 74.  
 Dash, 5-6, 20, 27, 29, 35, 54, 80.  
`\dashv` ( —| ), 178.  
`dd` (Didot point), 40.  
`\ddag` ( ‡ ), 61, 179.  
`\ddt` (debugging aid), 121, 130, 137, 142.  
`\def` (define a control sequence), 7, 16, 18, 33, 96-99, 112, 115, 118, 122, 127, 131, 136, 141, 143, 151-153.  
 Definition of control sequences, *see* `\def`.  
`\deg` (degree symbol), 154, 164.  
`\delete` (BS), 32, 165.  
`\delim`, 75, 79.  
`\Delta` (  $\Delta$  ), 177.  
`\delta` (  $\delta$  ), 61, 177.  
`\Deltait` (  $\Delta$  ), 177.  
 Demerits, 55-56.  
 Denominator, 66, 134.  
 Depth, 41-45.  
     of completed page, 59.  
`depth`, 99.  
`\det` (determinant operator), 72, 152.  
 Dick and Jane, 48.  
 Didot point, 40.  
 Digit-width space, 104, 165.  
`\dimen`, 41, 141.  
`\dimenparam`, 119, 128, 136.  
 Dimension parameters, 119, 128, 136.  
 Dimensions, 40-42.  
 Discretionary hyphen, 20, 54-55, 124, 143.  
 Discretionary times sign, 55, 85, 134.  
`\displayskip` (glue above display following short line), 92, 119, 128, 136, 153.  
`\displaybskip` (glue below display following short line), 92, 119, 128, 136, 153.  
`\display`, 125, 130.  
 Display break penalty, 120, 129.  
  
 Display math mode, 50-52.  
     summary, 130-138.  
 Display style, 65-67, 92.  
 Displayed formulas, 58, 91-96.  
`\displayskip` (glue above and below displays), 92, 119, 128, 136, 153.  
`\displaystyle` (display style), 68, 135.  
`\div` (  $\div$  ), 178.  
`\dleft` (  $\ll$  ), 178.  
`\doteq` (  $\doteq$  ), 10, 178.  
 Dotless letters, *see* `\l`, `\j`, `\litt`, `\jit`.  
 Dots, 85-87, 90-91.  
     ! Double . . . , 132, 140.  
`\down` (  $\downarrow$  ), 178.  
`\dright` (  $\gg$  ), 178.  
`\Dscr` (  $\mathscr{D}$  ), 177.  
 Dynamic programming, 55.  
  
 Editing on-line, 22, 147.  
`\eject` (force page break and line break), 24, 54-55, 57, 84, 112, 117, 126, 135.  
 Ellipses (three dots), 49, 85-87, 90-91.  
`\else` (begin false text of conditional input), 33, 111-114, 121, 130, 137, 140.  
 Em-dash, 5-6, 27, 29, 80.  
`\emptyset` (  $\emptyset$  ), 179.  
 En-dash, 5-6, 17, 35, 148, 155.  
`\end` (terminate TeX), 23, 52, 121, 140.  
 End of line, 28, 30-31, 33.  
 End of page in input file, 32, 141.  
 End of paragraph, 23, 28, 30-32, 51, 57, 114, 125, 135, 146.  
`\ENDV`, 118, 126-127.  
`\epsilon` (  $\epsilon$  ), 4, 61, 177.  
`\eqalign` (align equations), 92-96, 107, 152.  
`\eqalignno` (align numbered equations), 93-94, 107, 152.  
`\eqno` (displayed equation number), 91-94, 135-136.  
 Equation numbers, 91-94, 135.  
`\equiv` (  $\equiv$  ), 88, 152.  
 Error messages, 187, *see also* Error recovery.  
     complete list, 139-146.  
 Error recovery, 21-27, 114-115, 122, 131, 138-148.  
     errors.tmp, 24, 26, 147.  
 Escape character, 7, 18, 28, 30-32.

- Escape space, 8–9, 12, 33, 49, 81, 123, 133.  
`\Escr` (  $\mathcal{E}$  ), 177.  
`\eta` (  $\eta$  ), 177.  
 ex fonts, 73–74, 79, 118, 124, 125, 127, 134, 174–175.  
 Exception dictionary, 182–186.  
 Exclamation point, 48, 71, 82.  
 Exercises, 2.  
   answers, 148–151.  
`\exp` (exponential operator), 72–73, 152.  
 expand, 100, 104, 115, 124, 133.  
 Expansion of macros, 98, 106–107, 147.  
 Extension fonts, 73–74, 79, 118, 124, 125, 127, 134, 174–175.  
 Extensions to T<sub>E</sub>X, 117, 126, 135, 198.  
 ! Extra `\right`, 131, 140.  
 ! Extra `\}`, 115, 122, 131, 140.
- Factorial, 82, 91.  
 Feasible breaks in paragraph, 55.  
`ff`, see Ligatures.  
 File name, 25, 33, 121, 139.  
 Floating insertion, 59, 118, 127.  
`fmemsize`, 144.  
`\font`, 74, 118–119, 127–128, see also Font codes.  
 Font codes, 13–14, 33, 141.  
   table, 14.  
 Font definition, 12–15, 18, 36, 73–74, 115, 118–119, 127–128, 118–119, 127–128, 140, 145.  
 Font tables, 168–179.  
 Fonts, 12–15.  
`\footnote` (insert footnote), 13, 164, 167.  
 Footnotes, 12–13, 164.  
 Footnotes, 164.  
 for, 56, 120, 129.  
 Foreign language characters and accents, 37–39, 53–54.  
`\form-feed` (FF), 29, 32, 165.  
`\form-feed`, 12.  
`\format`, 104–106.  
`\formula`, 91, 125, 130.  
 Fractions, 64–69, 134. `\Fscr` (  $\mathcal{F}$  ), 17
- `\Gamma` (  $\Gamma$  ), 11, 61, 177.  
`\gamma` (  $\gamma$  ), 11, 61, 177.  
`\Gammaait` (  $\Gamma$  ), 177.  
`\gcd` (greatest common divisor operator), 72, 88, 152.  
`\gdef` (global `\def`), 18, 98, 112, 115, 118, 122, 127, 131, 136.  
`\glb` (  $\sqcap$  ), 178.  
 Glue, 41, 45–50, 54, 57–59, 80–81, 116, 125, 133.  
   above and below displays, 92.  
   above and below insertions, 59.  
   at top of page, 59, 153.  
   between lines, see Interline glue.  
   between paragraphs, 57.  
   between words, see Variable space.  
   between aligned columns or rows, 105–109.  
   setting of, 46, 99–103.  
 Glue parameter definitions, 115, 119, 128, 136.  
`\glueparam`, 119, 128, 136.  
 Grave accent, 38, 132, 148.  
`\grgr` (  $\gg$  ), 178.  
 Grouping, 13, 15–18, 28, 62–63, 78, 84, 98, 115, 122.  
   within groups, 15–16.  
`\Gscr` (  $\mathcal{G}$  ), 177.
- `\H` (long Hungarian umlaut), 38, 132.  
`\halign` (horizontal alignment), 52, 88–89, 92, 104–109, 117, 127, 135, 140.  
 ! `\halign` in display . . . , 95, 140.  
`\hangindent` (hanging indentation), 56, 120, 129.  
 Hanging indentation, 56, 120, 129.  
`hashsize`, 144.  
 Hat accent, 38, 64, 132.  
 Height, 41–45.  
   of completed page, 59.  
`height`, 99.  
`\hf111` (horizontal glue fill), 50, 69, 109, 113, 125, 133, 145.  
`\hjust` (horizontal justification), 52, 73, 93, 99–103, 105, 115, 124, 133, 145.  
`\hlist`, 99, 121.  
 Horizontal braces, 103.  
 Horizontal glue, 45–47, 50, 84, 125, 133.



- Horizontal list, 43-45, 99, 121.  
 Horizontal mode, 23, 50-52, 57.  
   summary, 121-130.  
 Horizontal rule, 43, 99, 101, 102, 105, 108-109, 115.  
`\hrule` (horizontal rule), 43, 99, 102, 108, 115, 148.  
`\Hscr` ( $\mathfrak{H}$ ), 177.  
`\hsize` (page width), 18, 20, 24-25, 50, 53, 50, 112-113, 119, 121, 128, 136, 150.  
`\hskip` (horizontal glue), 47, 50, 84, 125, 133.  
 Hyphen, 5-6, 54, *see also* Discretionary hyphen.  
 Hyphenation, 53-56, 180-186.  
 Hyphenation penalties, 54, 56, 120, 129.
- `\i` (dotless i), 39.  
`idlevs`, 144.  
`\if` (test equality of characters), 33, 111-114, 121, 130, 137, 140, 150.  
`\ifeven` (test parity of counter), 33, 111-113, 121, 130, 137, 140, 150.  
 Ignored character, 28, 30-31.  
 Ignore space, 31, 106, 112, 124, 133.  
`\iit` (dotless i), 179.  
`! Illegal` . . . , 141, 146.  
`\imag` ( $\mathfrak{I}$ ), 179.  
`in` (inch), 40-41.  
`\in` ( $\in$ ), 61, 178.  
 Indentation, 56, 116-117, *see also* Hanging indentation.  
`\inf` (infimum operator), 72, 152.  
 Infinite stretchability, 49-50.  
`\infty` ( $\infty$ ), 74, 179.  
`\input` (read specified file), 8, 19, 24-25, 121.  
 Insertions, into manuscript, 22.  
   into pages, 59, 118, 127.  
`\int` ( $\int$ ), 69, 81, 149, 179.  
 Integration, 69-70, 91.  
`\inter` ( $\cap$ ), 179.  
 Inter-word glue, *see* Variable space.  
`\interc` ( $\intercal$ ), 178.  
 Interline glue, 45, 58, 100, 109, 115.  
`\iota` ( $\iota$ ), 177.  
`\Isr` ( $\mathfrak{I}$ ), 177.  
`it` fonts, 73-74, 125, 172.  
`\it` (italic), 153.
- Italic correction, 43, 124, 132, 134, 142, 149.  
 Italic fonts, 73-74, 125, 172.  
 Italic letters, 74.
- `\j` (dotless j), 39.  
`\jit` (dotless j), 179.  
`\join` ( $\vee$ ), 179.  
 Jokes, 2.  
`\jpar` (justification feasibility parameter), 18, 25-26, 56, 120, 129, 143, 151.  
`\Jscr` ( $\mathfrak{J}$ ), 177.
- `\kappa` ( $\kappa$ ), 177.  
 Kerning, 6, 123.  
 Knuth, Donald E., 1, 10, 14, 155.  
`\Kscr` ( $\mathfrak{K}$ ), 177.
- `\l` (Polish accent), 38, 132, 148.  
`\Lambda` ( $\Lambda$ ), 177.  
`\lambda` ( $\lambda$ ), 88, 177.  
`\Lambdait` ( $\Lambda$ ), 177.  
`\langle` ( $\langle$ ), 75, 178.  
 Large delimiters, 74-79, 141.  
 Large operators, 70, 73, 77-78.  
`\lceil` ( $\lceil$ ), 75, 178.  
`\ldots` (. . . ), 49, 51, 80-87, 152.  
`\ldotsm` (`\ldots` in multiplication), 86-87, 149, 152.  
`\ldots` (`\ldots` followed by space), 86-87, 96-97, 152.  
 Leaders, 102-103, 116, 125, 139, 150.  
`\leaders` (leaders), 102-103, 116, 125, 139, 150.  
`\left` (variable-size left delimiter), 75-79, 82, 88-89, 90, 131.  
 Left brace, 75, 79, 88.  
`\leftset` (`{` in set definition), 88, 152.  
`\leftv` ( $|$ ), 87, 90, 178.  
`\leftvv` ( $||$ ), 87, 178.  
 Letter, 28, 32.  
`\letter`, 122, 131.  
`\lfloor` ( $\lfloor$ ), 75, 178.  
`\lft` (leftify), 89-90, 151.  
`\lg` (binary logarithm operator), 72, 151.  
 Liang, Frank M., 180.  
 Ligatures, 6, 10, 35, 123.  
`\lim` (limit operator), 72-73, 151.

- `\liminf` (inferior limit operator), 72, 151.  
 Limits to operators, 70–71, 73, 134.  
`\limitswitch` (change position of limits), 70, 134, 142, 151.  
`\limsup` (superior limit operator), 72, 151.  
 Line break penalties, 54, 56, 120, 126, 129, 135.  
 Line breaking, 25–26, 52–57, 71, 101.  
`\linefeed` (LF), 32, 165.  
`\linefeed`, 12.  
 Line rules, 43, 99, 101, 102, 105, 108–109, 115, 124.  
`\linoskip` (vertical glue between boxes with distant baselines), 58, 100, 109, 112, 115, 119, 128, 130, 149, 153.  
`\ln` (natural logarithm operator), 72, 151.  
`\log` (logarithm operator), 72–73, 151.  
`\lower` (shift a box down), 102, 125, 133.  
`\lpile` (left-justified pile), 90, 93, 152.  
`\Lscr` (  $\mathcal{L}$  ), 177.  
`\lscr` (  $\ell$  ), 177.  
`\lsh` (  $\uparrow$  ), 178.  
`\lsls` (  $\ll$  ), 178.  
`\lub` (  $\sqcup$  ), 178.  
  
 Macron accent, 38–39, 64, 132.  
 Macros, definition of, *see* `\def`.  
   expansion of, 98, 106–107, 147.  
   tracing, 147.  
   use of, 96–99, 115, 122, 131.  
`\mapsto` (  $\mapsto$  ), 61, 178.  
 Margins, *see* `\setsize`, `\hangindent`.  
`\mark` (define a mark), 33, 110–112, 117, 141.  
 Marks, 110–112, 117, 126, 135.  
 Math formulas, breaking, 54–55, 84–85, 94–96, 120, 129.  
 Math formulas, how to type, 60–96.  
 Math mode, 33, 49, 50–52, 57.  
   character conversion in, 35, 60, 176.  
   summary, 130–138.  
`\mathbin` (Bin box), 84, 132.  
`\mathchar`, 131–132.  
`\mathclose` (Close box), 84, 132.  
`\mathcontrol`, 132.  
`\mathex` (define *ex* font), 74, 118, 127, 153.  
`\mathglue`, 133.  
`\mathit` (define *it* font), 74, 119, 128, 153.  
  
`\mathop` (Op box), 84, 132.  
`\mathopen` (Open box), 84, 132.  
`\mathpunct` (Punct box), 84, 132.  
`\mathrel` (Rel box), 84, 132.  
`\mathrm` (define *rm* font), 74, 119, 128, 153.  
`\mathsf` (define *sy* font), 74, 119, 128, 153.  
`\mathstyle`, 135.  
 Matrices, 88–91, 104.  
`\max` (maximum operator), 72–73, 83, 151.  
`\maxdepth` (maximum page depth), 18, 59–60, 112–113, 119, 128, 136, 153.  
`\meet` (  $\bigwedge$  ), 179.  
`memsize`, 144.  
 Metric units, 40.  
`\min` (minimum operator), 72, 83, 151.  
 minus, 47, 50.  
 Minus sign, 5–6, 36, 60.  
 ! Missing `...`, 21, 142.  
 ! Missing `\cr` `...`, 115, 122, 142.  
 ! Missing `\right`, 131, 142.  
 ! Missing `$` `...`, 135, 142.  
 ! Missing `{`, 138, 142.  
 ! Missing `}`, 118, 127, 131, 142.  
`\mlist`, 130.  
 mm (millimeter), 40.  
`\mod` (modulus operator), 88, 152.  
 Modes, 23, 29, 50–52, 140.  
`\modop` (mod operator), 88, 149, 152, 154.  
`\moveleft` (shift a box left), 102, 116.  
`\moveright` (shift a box right), 102, 116.  
`\mp` (  $\mp$  ), 178.  
`\Mscr` (  $\mathcal{M}$  ), 149, 177.  
`\mu` (  $\mu$  ), 177.  
`\munion` (  $\cup$  ), 179.  
  
`\nabla` (  $\nabla$  ), 179.  
 Narrow margins, 26, 53.  
 Natural width of list, 40, 48, 101, 105.  
 Negative dimensions, 44–45.  
 Negative glue, 50, 81, 109.  
`\neq` (  $\neq$  ), 88, 152, 178.  
`nestsize`, 144.  
 Newspapers, 26.  
`\noalign` (disable alignment), 33, 93, 105, 107–108, 118, 127.  
`\noindent` (begin nonindented paragraph), 56, 112, 116.

- $\langle$ nonmathletter $\rangle$ , 122.  
 $\backslash$ not (cancel relation), 152, 178.  
 $\backslash$ notin ( $\notin$ ), 178.  
 $\backslash$ Nscr ( $\mathcal{N}$ ), 177.  
 $\backslash$ nu ( $\nu$ ), 177.  
 $\backslash$ null (empty box of size zero), 59, 95-90, 107, 149, 152, 153.  
 $\langle$ null $\rangle$  (NUL), 32, 165.  
 $\langle$ number $\rangle$ , 33, 34, 40-41.  
 Number theory formulas, 88, 154.  
 Numerator, 66, 134.  
  
 O vs. o, 73, 81-82.  
 $\backslash$ O ( $\emptyset$ ), 37, 122, 148.  
 $\backslash$ o ( $\phi$ ), 37, 122.  
 Octal notation, 34.  
 $\backslash$ odiv ( $\oslash$ ), 178.  
 $\backslash$ odot ( $\odot$ ), 178.  
 $\backslash$ odotprod ( $\odot$ ), 179.  
 $\backslash$ OE ( $\mathbb{O}$ ), 37, 122.  
 $\backslash$ oe ( $\alpha$ ), 37, 54, 122.  
 $\backslash$ oint ( $\oint$ ), 179.  
 $\backslash$ Omega ( $\Omega$ ), 177.  
 $\backslash$ omega ( $\omega$ ), 61, 177.  
 $\backslash$ Omegait ( $\Omega$ ), 177.  
 $\backslash$ ominus ( $\ominus$ ), 178.  
 On-line editing, 22, 147.  
 Op box, 82-84, 132, 134, 142, 176.  
 Open box, 82-84, 133, 176.  
 $\backslash$ oplus ( $\oplus$ ), 10, 178.  
 $\backslash$ oprod ( $\otimes$ ), 179.  
 $\langle$ optional sign $\rangle$ , 40.  
 Ord box, 82-84, 132, 176.  
 Organs, 29.  
 $\backslash$ Oscr ( $\mathcal{O}$ ), 177.  
 $\backslash$ osum ( $\oplus$ ), 179.  
 Other character, 28, 32.  
 $\langle$ otherchar $\rangle$ , 28, 32, 122, 131.  
 $\backslash$ otimes ( $\otimes$ ), 178.  
 $\backslash$ output (define output routine), 18, 33, 59-60, 98, 104, 109-114, 120, 129, 137, 141, 147, 153, 166.  
 Output routines, see  $\backslash$ output.  
 $\backslash$ over (specify built-up fraction), 64, 82, 130, 134, 139, 149.  
 $\backslash$ overline (put line over formula), 63, 67, 82, 132, 134.  
  
 Overfull box, 25-26, 143, 147.  
  
 $\backslash$ P ( $\P$ ), 61, 179.  
 $\backslash$ page (completed page), 60, 102, 111, 113, 115-116, 124, 133, 143.  
 Page break, 57-60.  
 Page break penalties, 58-59, 117, 120, 129.  
 Page building, 57-60.  
 Page numbers, 59-60, 111-113.  
 $\backslash$ par (end of paragraph), 23, 28, 30-32, 51, 57, 114, 125, 135, 146.  
 Paragraph, beginning of, 56, 116-117.  
     ending of, see  $\backslash$ par.  
 Parameter, 97.  
 Parameter text, 97-98.  
 Parameters to T<sub>E</sub>X, see  $\backslash$ chpar, Glue parameters, Dimension parameters, Size parameters.  
 Parentheses, 42, 75.  
 $\backslash$ parindent (amount of indentation at beginning of paragraph), 18, 56, 117, 119, 128, 136, 153.  
 parsize, 144.  
 $\backslash$ parskip (additional glue between paragraphs), 57, 116, 119, 128, 136, 153.  
 $\backslash$ partial ( $\partial$ ), 149, 179.  
 pc (pica), 40.  
 Penalties for hyphenation, 54, 56, 120, 129.  
 Penalties, 27, 54-59, 64, 105, 107, 117, 120, 126, 129, 135.  
 $\backslash$ penalty (line or page break penalty), 27, 54-57, 64, 107, 117, 126, 135.  
 Period, 48-49, 71-72, 82.  
 $\backslash$ perp ( $\perp$ ), 178.  
 $\backslash$ Phi ( $\Phi$ ), 177.  
 $\backslash$ phi ( $\phi$ ), 61, 177.  
 $\backslash$ Phit ( $\Phi$ ), 177.  
 $\backslash$ Pi ( $\Pi$ ), 177.  
 $\backslash$ pi ( $\pi$ ), 52, 149, 177.  
 Pica, 40.  
 $\backslash$ Pit ( $\Pi$ ), 177.  
 plus, 47, 50.  
 $\backslash$ pm ( $\pm$ ), 178.  
 Point, 40-41.  
 Polish crossed l and L, 38-39.  
 $\backslash$ Pr (probability operator), 72, 152.  
 Preamble to an alignment, 104-109.




- $\backslash$ prec ( $\prec$ ), 178.  
 $\backslash$ preceq ( $\preceq$ ), 178.  
 Pretzels, 79.  
 $\backslash$ prime ( $\prime$ ), 63, 179.  
 $\backslash$ prod ( $\prod$ ), 179.  
 Proper names, 53.  
 $\backslash$ Pscr ( $\mathcal{P}$ ), 149, 177.  
 $\backslash$ Psi ( $\Psi$ ), 177.  
 $\backslash$ psi ( $\psi$ ), 177.  
 $\backslash$ Psiit ( $\Psi$ ), 177.  
 pt (point), 40, 78, 141.  
 Punct box, 82–84, 133, 176.  
 Punctuation marks, 48, 53.  
     in math formulas, 71–72.
- $\backslash$ quad (double quad space), 80, 89, 152.  
 $\backslash$ quad (quad space), 80–81, 88–89, 123, 133.  
 Quad middle, 57.  
 Quads, 80.  
 Question mark, 48, 71.  
 Quotation marks, 4–7, 20, 35, 48.  
 $\backslash$ Qscr ( $\mathcal{Q}$ ), 177.
- $\backslash$ ragged (degree of raggedness), 25–26, 50,  
     120, 129, 151.  
 Ragged right margins, 26, 50.  
 $\backslash$ raise (shift a box up), 63, 102, 125, 133,  
     139.  
 $\backslash$ rangle ( $\rangle$ ), 75, 178.  
 $\backslash$ rcell ( $\rangle$ ), 75, 178.  
 $\backslash$ real ( $\mathbb{R}$ ), 179.  
 Recovery from errors, 21–27, 114–115, 122,  
     131, 138–148.  
 Reference point, 42–45.  
 Rel box, 82–85, 133, 176.  
 Relations, 55–56, 82–83, 96, 178.  
 $\backslash$ relv ( $\rangle$ ), 87–88, 178.  
 $\backslash$ relvv ( $\rangle\rangle$ ), 87, 178.  
 Restricted horizontal mode, 50, 52.  
     summary, 121–130.  
 Restricted vertical mode, 33, 50, 52.  
     summary, 114–121.  
 Result text, 97–99.  
 $\backslash$ rfloor ( $\rfloor$ ), 75, 178.  
 $\backslash$ rho ( $\rho$ ), 177.  
 $\backslash$ right (variable-size right delimiter), 75–79,  
     82, 88–89, 96, 131.
- Right brace, 88.  
 Right justification, 50, 151.  
 $\backslash$ rightsot ( $\}$  in set definition), 88, 152.  
 $\backslash$ rightv ( $\rangle$ ), 87, 90, 178.  
 $\backslash$ rightvv ( $\rangle\rangle$ ), 87, 178.  
 $\backslash$ rjustline (right justify a line), 151.  
 rm fonts, 73–74, 125, 170.  
 $\backslash$ rm (roman), 12, 153.  
 Roman fonts, 170.  
 Roman letters in formulas, 72–74.  
 Roots, 63.  
 $\backslash$ rpile (right-justified pile), 90, 152.  
 $\backslash$ Rscr ( $\mathcal{R}$ ), 177.  
 $\backslash$ rsh ( $\rsh$ ), 178.  
 $\backslash$ relash ( $\rangle$ ), 178.  
 $\backslash$ rt (rightify), 89, 151.  
 Rule box (line rule), 43, 99, 101, 102, 105,  
     108–109, 115, 124.  
 Rulers, 41.  
 Runaway argument, 141.  
 Running headline, 112, 155.  
 Running TeX, 18–27.
- $\backslash$ s (tilde), 38, 64, 132.  
 $\backslash$ save (save a box), 101, 113–114, 116, 125,  
     133, 139.  
 savesize, 144.  
 Scandinavian circle accent, 38–39, 132.  
 Script letters, 10, 177.  
 Script size, 65, 74.  
 Script style, 65–67.  
 Scriptscript size, 66, 74.  
 Scriptscript style, 65–67.  
 $\backslash$ scriptscriptstyle (scriptscript style),  
     63, 68, 135.  
 $\backslash$ scriptstyle (script style), 68, 71, 135, 149.  
 $\backslash$ sec (secant operator), 72, 151.  
 $\backslash$ section ( $\S$ ), 61, 179.  
 Semicolon, 48, 71, 82.  
 $\backslash$ setcount (set a counter), 111, 120, 129, 137.  
 Sets in math, 61, 88.  
 Setting glue, 46, 99–103.  
 Shifted boxes, 102, 116, 125, 133.  
 Shrink component of glue, 45–50, 55, 58, 101.  
 $\backslash$ Sigma ( $\Sigma$ ), 177.  
 $\backslash$ sigma ( $\sigma$ ), 177.  
 $\backslash$ Sigmait ( $\Sigma$ ), 177.

- `\simeq` ( $\approx$ ), 178.
- `\sin` (sine operator), 72-73, 151.
- `size`, 33, 100, 104, 115, 124, 133.
- `\sl` (slanted roman), 12, 17, 153.
- Slanted type, 157.
- Slash, 36, 82.
- Slavic accent, 38, 132.
- `\smallint` ( $\int$ ), 179.
- Space, see Blank space character, Glue, Variable space.
- `\space`, 114, 121, 123, 124, 131.
- Space component of glue, 45-50.
- Space factor, 48, 53.
- Spacing in math formulas, 79-84, 87-88.
  - table, 81.
- `\spec`, 33, 115, 124, 133.
- `\superpose` (superpose), 39, 63, 101, 106-109, 152.
- `\sqrt` (square root), 63, 67, 74-75, 78, 81-82, 132, 134.
- `\sqsub` ( $\sqsubseteq$ ), 178.
- Square root, see `\sqrt`.
- `\squnion` ( $\sqcup$ ), 179.
- `\ss` ( $\mathcal{B}$ ), 37, 122, 148.
- `\Sscr` ( $\mathcal{J}$ ), 177.
- `stacksize`, 144.
- Stanford conventions, 19, 24, 198.
- States, 29-33.
- `\sterling` ( $\pounds$ ), 61, 179.
- Stopping TeX, 22, 121.
- Stretch component of glue, 45-50, 55.
  - infinite, 49-50.
- `stringize`, 144.
- Styles of math setting, 65-67, 130, 135.
- SUAI code, 18, 35, 169.
- Subformula, 82, 130-131.
- Subscript, 62-63, 66, 84, 132, 134, 140.
- `\subset` ( $\subseteq$ ), 178.
- `\succ` ( $\succ$ ), 178.
- `\succeq` ( $\succeq$ ), 178.
- `\sum` ( $\sum$ ), 64, 69, 77-78, 96, 149, 179.
- Summation, 69-71, 91.
- `\sup` (supremum operator), 72, 152.
- Superposition, 63, 101, 106-109, 152.
- Superscripts, 62-63, 66, 78, 84, 132, 134, 140.
- `\supset` ( $\supseteq$ ), 178.
- `\surd` ( $\checkmark$ ), 179.
- sy fonts, 73-74, 79, 125, 173.
- Symbol fonts, 73-74, 79, 125, 173.
- `\t` ( $\tau$ ), 36, 132, 148.
- `\tab` (TAB), 32, 165.
- `\{tab`, 12.
- `\tabskip` (glue in alignments), 18, 105-109, 118, 119, 127, 128, 136.
- `\tan` (tangent operator), 72, 151.
- `\tau` ( $\tau$ ), 4, 177.
- `\TeX` (TeX logo), 9, 17.
- TeX, meaning of, 4.
- TeX logo, 9, 17, 44.
- Text size, 66, 74.
- Text style, 65-67.
- `\textindent` (insert text into paragraph indent), 159, 165.
- `\textstyle` (text style), 68, 135.
- .TFX, 198.
- Theorems, 157-158.
- ! There's no ..., 117, 126, 135, 144.
- `\Theta` ( $\Theta$ ), 177.
- `\theta` ( $\theta$ ), 61, 73, 177.
- `\thetait` ( $\Theta$ ), 177.
- Thick space, 80-84.
- Thin space, 7, 71, 80-84, 88, 133, 151-152.
- Three-column format, 113-114.
- Three dots, 85-87, 90-91.
- Tie accent, 38, 132, 148.
- Tilde accent, 38, 64, 132.
- `\times` ( $\times$ ), 178.
- to, 33.
- to (dimen), 101, 104, 115, 124, 133, 145.
- to size, 100, 104, 115, 124, 133.
- Tokens, 22, 97-98, 146.
- ! Too many ..., 115, 122, 145.
- `\top` ( $\top$ ), 179.
- `\topbaseline` (normal position of top baseline on page), 18, 59, 119, 128, 136, 153.
- `\topinsert` (insertion at top of page), 52, 54, 57, 59, 118, 127.
- `\topmark` (current mark at top of page), 110-112, 117, 126, 135, 166.
- `\topskip` (glue below `\topinsert`), 59, 119, 128, 136.

- `\traco` (combination of tracing facilities), 18, 98, 120, 121, 129, 130, 137, 142, 147, 151.  
 Tracing, 140-147.  
 Translation of characters in math mode, 35, 60, 170.  
 Truth, 2.  
`\Tscr` (  $\mathcal{T}$  ), 177.  
 Two-column format, 113-114.  
`\twoline` (two line display), 94-96, 152.  
 Typewriter fonts, 171.  
 Typewriter type used in this manual, 5.
- `\u` (breve), 38-39, 132.  
 Umlaut accent, 8, 23, 35, 38, 132, 148.  
 ! `Undefined . . .`, 21, 114, 122, 131, 145.  
`\underlino` (put line under formula), 11, 63, 67, 82, 132, 134.  
 Underlining, 162, *see also* `\underlino`.  
`\union` (  $\cup$  ), 179.  
 Units of measure, 33, 40-41.  
`\up` (  $\uparrow$  ), 178.  
`\uplus` (  $\uplus$  ), 178.  
`\Upsilon` (  $\Upsilon$  ), 177.  
`\upsilon` (  $\upsilon$  ), 177.  
`\Upsilonnit` (  $\Upsilon$  ), 177.  
`\Uscr` (  $\mathcal{U}$  ), 177.  
 ! Use of . . . , 97, 145.
- `\v` (Slavic accent), 38, 132.  
`\valign` (vertical alignment), 52, 104, 109, 115, 118, 120.  
 Variable-size delimiters, 74-79.  
 Variable space, 26, 48.  
`\varomega` (  $\varpi$  ), 61, 177.  
`\varphi` (  $\varphi$  ), 61, 177.  
`varsize`, 144.  
`\vartheta` (  $\vartheta$  ), 61, 177.  
`\vcenter` (vertically center a `\vlist` box), 88-89, 93, 102, 107, 135.  
`\vdash` (  $\vdash$  ), 178.  
`\vdots` (vertical ellipsis), 90-91, 152.  
 Vector, 85-86, 91, 96-97.  
 Vector accent, 38, 64, 132.  
 Vertical glue, 20, 40, 47, 58, 105, 112, 110, 140.  
 Vertical lines in math, 75, 87.  
 Vertical list, 43-45, 114.
- Vertical mode, 33, 50-52, 57.  
     summary, 114-121.  
 Vertical rules, 43, 99, 101, 102, 108-109, 124.  
`\vertical-tab`, 12.  
`\vertical-tab` (VT), 32, 165.  
`\vfill` (vertical glue fill), 23, 50, 52, 58, 112-113, 116.  
`\vjust` (vertical justification), 52, 100-103, 107, 112-113, 115, 124, 133.  
`\vlist`, 100, 105, 114.  
 Vowel, 181.  
`\vrule` (vertical rule), 43, 99, 108, 124, 148.  
`\Vscr` (  $\mathcal{V}$  ), 177.  
`\vsize` (page height), 18, 59, 112-113, 119, 121, 128, 130.  
`\vskip` (vertical glue), 20, 40, 47, 58, 112, 116, 140.  
`\vtop` (make `\vlist` box using top baseline), 102, 135.
- Warning: . . . , 198.  
 Widow lines, penalty for, 58-59, 120, 129.  
 Width, 41-45.  
     width, 99.  
`\wp` (  $\wp$  ), 179.  
`\Wscr` (  $\mathcal{W}$  ), 177.
- `\x` (extension), 117, 126, 135.  
 XGP, 20, 24, 198.  
`xgp`, 198.  
`\xi` (  $\xi$  ), 177.  
`\xi` (  $\xi$  ), 177.  
`\xiit` (  $\xi$  ), 177.  
`\Xscr` (  $\mathcal{X}$  ), 177.  
`\xskip` (additional space in text), 47, 158-160, 165.
- ! You can't . . . , 121, 130, 138, 146.  
`\Yscr` (  $\mathcal{Y}$  ), 177.  
`\yskip` (extra space between paragraphs), 47, 159, 165.  
`\yyskip` (double `\yskip`), 47, 159, 165.
- `\zeta` (  $\zeta$  ), 177.  
`\Zscr` (  $\mathcal{Z}$  ), 10, 177.

- 7-bit character codes, 32, 34-35, 122, 168-173.  
 \9 (digit-width space), 164, 165.  
 9-bit character codes, 74, 79, 132, 141, 176.
- ( ), (SP), 32, 165.  
 □, 5.  
 \□, 8-9, 12, 33, 49, 81, 123, 133.  
 \! (negative thin space or ignore space), 31, 81, 106, 112, 124, 133.  
 \~ (umlaut), 8, 23, 35, 38, 132, 148.  
 ‡, 28, 97-98, 104-109, 142.  
 \# ( # ), 179.  
 ‡‡, 97-99.  
 \$, 28, 29, 51, 60, 66, 125, 131.  
 \\$ ( \$ ), 61, 108, 179.  
 \$\$, 33, 51-52, 60, 125-126.  
 %, 28, 34.  
 \% ( % ), 34, 151.  
 \^ (grave accent), 38, 132, 148.  
 \' (acute accent), 8, 10, 38, 132.  
 \\* (discretionary ×), 55, 84, 85, 134.  
 \. (thin space), 7, 81-84, 88, 133, 151-152.  
 -, 36, 123.  
 \- (discretionary hyphen), 26, 54, 124.  
 /, 36, 82.  
 \/ (italic correction), 43, 124, 142.  
 \, 7, 28.  
 \: (select current font), 12-15, 18, 36, 118-119, 127-128, 130.  
 \; (thick space), 81, 83-84, 133.  
 \< (negative op space), 81, 133, 152.  
 \≤ (negative conditional thin space), 81, 133.  
 \= (macron), 38-39, 64, 132.  
 \≥ (conditional thin space), 81, 83, 133, 152.  
 \> (op space), 81, 83, 133.  
 \? (negative thick space), 81, 133.  
 {, 15-16, 28, 62-63, 115, 121, 122, 131.  
 \{ ( { ), 75, 90, 178.  
 }, 15-16, 28, 62-63, 115, 121, 122, 131, 141.  
 \} ( } ), 75, 178.  
 †, 28, 62-63, 70, 132, 140.  
 \† ( † ), 178.  
 ‡, 28, 62-63, 70, 73, 132, 140.  
 \‡ ( ‡ ), 178.  
 †, 179.  
 \← ( ← ), 10, 178.  
 →, 178.  
 \→ ( → ), 178.  
 \↔ ( ↔ ), 178.  
 \⊙ ( ⊙ ), 179.  
 α, 35.  
 \| ( || ), 75, 179.  
 ⊙, 28, 88-89, 104-108, 117-118, 126-127, 135, 144.  
 ..., 49, 85-87, 90-91.

**<S> Special notes about using T<sub>E</sub>X at Stanford**

- (1) The standard T<sub>E</sub>X program that you get by typing "r tox" requires that fonts @, a, d, f, g, j, l, n, q, u, x, z, and ? be reserved for the fonts declared in Appendix B. (The reason is that the system program already has the font information for these fonts in its memory; this avoids making T<sub>E</sub>X reload thirteen separate font information files each time.)
- (2) The standard T<sub>E</sub>X program produces output for the XGP. To produce output for the Alphatype (when it is available) we will use another program "texa".
- (3) You can type "xgp" before a unit of measure, to avoid the expansion factor. For example, "\hsize 3 xgpin" gives 3 X 200 pixels, which equals 3 inches (more or less) on our XGP
- (4) The extension ".TEX" is assumed to apply to \input file names if you do not specify the extension. If T<sub>E</sub>X can't find the file in your area, it tries system area [1, 3] before giving up. (File basic.TEX is on this area.) Your output file will have the same name as the first file you \input, except that the extension will be changed to "XGP".
- (5) The message "Warning: page limits exceeded!" is given when you try to output something below the place where the output page is cut, i.e., more than one xgp inch below the bottom of the box output by the \output routine.
-  (6) If a font you are using isn't on area [XGP, SYS], you must mention the area explicitly. T<sub>E</sub>X ignores the extension on font file names; the XGP server will assume that the extension is ".FNT", and T<sub>E</sub>X assumes that the font information is on another file with the extension ".TFX". (This applies to XGP fonts only; Alphatype fonts will be on area [ALP, SYS], and the corresponding T<sub>E</sub>X font information will have extension ".TFA".)
-  (7) Documentation for the T<sub>E</sub>X processor appears in the file TEXSYS.SAI on area [TEX, DEK], and in several other files mentioned there.
-  (8) The implementation of T<sub>E</sub>X is explicitly designed so that extensions can be written in SAIL and incorporated into your private version of the system. You write a module called TEXEXT.SAI and this replaces the dummy extension module that is ordinarily loaded with the T<sub>E</sub>X processor.