Do not remove from C-28

A Guide to TEX for the Troff User

Mike Urban
TRW Software Productivity Project

February 3, 1984

Abstract

Donald Knuth's TEX is a text processing system with many powerful and convenient features. Its architecture and user interface are substantially different from the *troff* family of programs with which most UNIX™ users are familiar. An introduction to the archiTEXture of Knuth's system is presented, with many references to the corresponding features of *troff*. This version corresponds to pre-release Version 0.9999 of TEX.

Table of Contents

A Guide to TEX for the Troff User

Mike Urban
TRW Software Productivity Project

1. Introduction

Donald Knuth's TEX system has several advantages over the standard *troff* program distributed with Berkeley or Bell UNIX[1]. TEX is portable, as it is written in Pascal, and is available on a variety of systems such as UNIX, VMS, TOPS-20[2], and IBM's VM/CMS[3] system. It is inexpensive ($50 for the UNIX port from the University of Washington), and it generates device-independent output. It also can produce beautiful copy, especially when setting mathematics. Finally, it has arbitrary-length command and variable names, which can provide much greater readability than the 1- and 2-character symbols of *troff* (this property is shared with the Scribe™ system from Unilogic, Inc.). For example, the first few lines of the file which produced this paper are:

```
\input paperhead    % Read private macro package
\centerline{\titlefont A Guide to \TeX}
\vskip 4pt
\centerline{\titlefont for the Troff User}
\vskip 8pt
\centerline{\it Mike Urban}
\centerline{\sl TRW Software Productivity Project}
\vskip .5in
\Section{Introduction}
Donald Knuth's \TeX\ system has several advantages over the
```

The only cryptic item in this excerpt is the \vskip primitive, which stands for **Vertical Skip**; the remaining commands are fairly transparent.

TEX is, however, substantially different in its general architecture from *troff*. Its input is "stream-oriented" rather than consisting of discrete lines, with embedded formatting commands preceded by an escape, or distinguished character (the backslash, \). Its treatment of different typefaces is different from the approach used by *troff*, which was designed to drive a particular (and now somewhat obsolete) variety of typesetter. Finally, TEX sets up a page of type using an algorithm based on "gluing" together "boxes" of characters, in a manner quite analogous to the way that traditional printers locked up "composing sticks" containing rows of letters by adjusting external wedges called "quoins." To the casual user setting up a simple document, the difference may appear to be minor. However, the consequences can be felt when one is attempting an operation that is even slightly extraordinary.

For these reasons, using TEX for the first time may prove to be a strange, perhaps even frustrating experience for an experienced *troff* user, even if a wealth of "user-friendly" macros (analogous to the –mm or –me macro packages) has been provided. This document will attempt to assist the prospective TEX user in the transition from *troff*. It does not pretend to be a complete description of TEX, nor is it a "cookbook" of conversion techniques for *troff* documents. It is simply an introduction to the most basic elements of TEX that assumes some *troff* experience. When in doubt, the reader is urged to try out the various features of TEX to see how they behave in practice, rather than relying on "thought experiments" based on the material in this paper or "*The TEXbook*".

---

[1] UNIX is a Registered Trademark of Bell Laboratories. Some say that's the problem.

[2] VMS and TOPS-20 are Trademarks of Digital Equipment Corporation

[3] VM/CMS is a Trademark of IBM

2. Input Format

2.1. *Control Sequences*

The *troff* input format with which the reader is familiar consists of lines of text. Some of these lines begin with a command character (i.e. a period) and are considered command lines; these contain formatting directives. The remaining lines contain text to be formatted, with occasional escape sequences (using a distinguished, or *escape* character) for particular formatting operations that cannot be easily specified at a line-break (such as in the middle of a word), or for special characters such as the Greek letter 'α' (`\(*a`).

In contrast, TEX input is "stream-oriented", i.e. the boundaries between input lines are unimportant. During normal operations, there is no substantial difference between a newline and a space (there are two exceptions to be noted later). TEX commands (called *control sequences*) may occur anywhere in the input, and are distinguished by beginning with an *escape* character (\. the backslash)[4]. There are two varieties of control sequences. The first is called a *control word* and consists of the escape character followed by one or more alphabetic characters. The end of a control word is the first non-alphabetic character encountered. Thus the control sequence `\TeX`, which sets the word 'TEX,' could be used in a phrase like `\TeX82` (which produces 'TEX82'), because the non-alphabetic character '8' signifies the end of the `\TeX` control word. However, the phrase `\TeXnical` would be considered an undefined control word.

The second kind of control sequence, like `\'`, is called a *control symbol*; it consists of the escape character followed by a *single nonletter*. For example, the escape sequence `\'` is used to accent letters: the input `P\'olya` yields 'Pólya'.

When a space follows a control word, it is ignored. For example, to set the word 'TEXnical' into type, one would type the input `\TeX nical`: the space following the `\TeX` control word ends the control word, but is otherwise ignored. Multiple spaces are treated as a single space. So to set the phrase 'TEX ignores spaces after control words' one must say:

```
\TeX\ ignores spaces after control words
```

using the control symbol `\␣` (i.e. an escape followed by a space) to set an actual space.

There are about 300 *primitive* control sequences built into TEX, and 600 more *defined* control sequences (many of them are *macros*) are supplied as part of the default version of TEX. These 900 definitions constitute "*plain* TEX." Plain TEX provides substantially more built-in support than *troff* does without macros, but does not provide all the facilities (section numbering, tables of contents) of a typical *troff* macro package. Thus, additional macros (probably supplied in a macro package) are usually required to format a complex document, but simple documents with unnumbered paragraphs, and even with footnotes and page headers, often can be formatted using only the facilities of plain TEX.

2.2. *Input lines*

As mentioned above, there are two exceptions to the rule whereby a newline is treated the same as a space. The first case is *comments*. When TEX encounters a percent sign (`%`), the percent sign, and all following characters up to and including the next newline, are ignored (one can set a percent in a document by using the control symbol `\%`).

The second case is *paragraphs*. When two consecutive newlines are encountered (i.e. an empty line), the effect is the same as if a `\par` control word, which represents an end-of-paragraph command, has occurred. Thus, a user can set up a simple document quite easily by simply leaving empty lines between paragraphs.

One can also use a plain TEX macro, `\obeylines`, to cause each newline character to begin a new paragraph, just as two consecutive newlines normally do. The effect is similar to the `.na` request of *troff*. However, as there is no macro to undo this effect, this should be done inside a *group*, to be described later.

[4] This character has nothing to do with the ASCII 'ESC' character (character code 036_8) which is labeled ESC on many terminals.

2.3. *Other Strange Characters*

Besides \ and %, there are eight other characters that have special significance for TEX, and should be avoided. If one must type them, there are control sequences that can be used to prevent their special meanings:

Character	*Name*	*Significance*	*Control Sequence*
\	Backslash	Escape	`$\backslash$`
{	Left Brace	Begin Group	`$\{$`
}	Right Brace	End Group	`$\}$`
$	Dollar Sign	Math Escape	`\$`
&	Ampersand	Alignment Tab	`\&`
#	Pound Sign	Parameter	`\#`
^	Circumflex	Superscript	`\^` *(accent)*
_	Underscore	Subscript	`\_`
~	Tilde	Tie	`\~` *(accent)*
%	Percent	Comment	`\%`

2.4. *Groups*

The characters { and } deserve special attention. These braces delimit *groups*. Groups correspond, very roughly, to *troff* environments (which most users don't see except in the effects of such macros as the .(b macro of the **–me** package). Basically, any formatting change that occurs within a group is undone when that group is closed. For example, the plain TEX macro `\bf` changes the current font to **bold face**; to change one word to **bold** in the middle of the sentence, one says something like

```
to change one word to {\bf bold} in the middle
```

The change to boldface within the group is undone when that group is closed.

Groups also serve a second function: they delimit parameters for certain control sequences. For example, the plain TEX macro `\centerline` requires a parameter comprising the text to be centered. Thus, the first title line of this document was produced by the input

```
\centerline{\titlefont A Guide to \TeX}
```

as seen in the introduction. Groups are not precisely analogous to anything in *troff*, but are an extremely important element of TEX syntax.

3. Fonts and Sizes

3.1. *Fonts*

Troff is oriented towards a C/A/T Phototypesetter[5], and this causes several peculiarities and limitations, such as the four-font restriction and the limitation of page width to 7.54 inches. Furthermore, the phototypesetter produces different sizes of type by magnifying the letters with lenses. This causes the smaller point sizes to be cramped and hard to read, while the much larger sizes can be more widely spaced than æsthetics might dictate.

TEX produces device-independent output files with a theoretical resolution smaller than the wavelength of visible light (65536 basic units to a printer's point). Characters on the page are symbols (often called *glyphs*) from particular fonts. The size of the symbol is defined by the font in which it occurs, just as its shape is. Thus, ten-point roman (which you are reading now) is a font completely separate from eight-point

[5] A device-independent *troff* is available as a separate program product from Western Electric or maybe some other AT&T subsidiary.

roman (which you are reading now). Plain TEX assumes that the document will be set in ten-point type, and sets the distance between lines to account for that. Plain TEX also defines five control sequences for changing fonts:

`\rm` switches to the normal "roman" typeface:	Roman
`\sl` switches to a slanted roman typeface:	*Slanted*
`\it` switches to italic style:	*Italic*
`\tt` switches to a typewriter-like face:	`Typewriter`
`\bf` switches to an extended boldface style:	**Bold**

At the beginning of a run you get roman type. As we saw earlier, the best way to switch fonts is within a group; this obviates the need for explicitly switching back to the previous font. Notice that these control sequences all change to the ten-point version of these type styles, regardless of the point size currently being used.

Users of *troff*, who are accustomed to using *italics* for emphasis and special material, should note carefully the difference between italics and *slanted letters*. *Slanted letters are just skewed roman letters, but italic letters are drawn in a completely different style.* Consider, for example, unslanted italic letters. Usually, slanted letters can be used for emphasis or for cited book titles; italics can be used for special material such as section headings, or names of UNIX programs such as *troff*. However, there are no established conventions for the use of italics versus slanted letters, and the writer is free to use them as he or she sees fit.

In addition to the five plain TEX fonts, one can easily define a control sequence which will switch to any single font in the font library stored in the UNIX file system (which normally includes the Computer Modern font family you are now reading). In effect, one is creating an association between a font name, used as a control sequence within TEX, and the name of the file on disk where the information about the font can be found. For example, the word "UNIX" which appears throughout this document is set in a ten-point small-caps font. The UNIX file name for this font is `amcsc10`[6]. The UNIX file names have no intrinsic significance to TEX, so to use `amcsc10` in this paper, a `\csc` control sequence was defined and associated with the `amcsc10` disk file by saying

```
\font\csc=amcsc10
```

After the `\font` control sequence, a switch to title font is accomplished by simply issuing the `\csc` control sequence. Samples of some of the available fonts have been appended to this paper.

Plain TEX provides some control sequences for smaller character sizes. For example, `\sevenbf` selects **seven-point boldface** type. When switching to a different size of type, one should be aware that the interline spacing is *not* changed, nor are the definitions for `\sl`, `\bf`, etc. changed; they still refer to ten-point slanted, boldface, etc.[7] To use the other available character sizes, such as nine-point roman, one must use `\font` to associate a new control word (e.g. `\ninerm`) with the appropriate font information file (in this case, `amr9`).

When switching from a slanty font like *italics* to a vertical one like this roman font, a little extra space should be inserted to prevent the top of the slanty text from getting too close to the unslanted material. Otherwise, it ends up '*printed* like this' instead of '*printed* like this.' This so-called *italic correction* should be set with the control symbol `\/` in the slanty font whenever such a switch is made. For example, the previous sentence began:

```
This so-called {\sl italic correction\/} should be set
```

This isn't too hard to get used to, and can prevent some uglinesses that *troff* doesn't understand or attempt to deal with.

---

[6] At present, the fonts have names like `amxxx`, standing for "Almost Computer Modern". It is expected that these names will progress through `bmxxx` (Becoming Computer Modern) and finally reach `cmxxx`.

[7] The macro package used to set this paper includes an `\eightpoint` macro that changes all the appropriate definitions to eight-point type, as in this footnote. See your TEX guru to see if such a macro is publicly available.

3.2. *Magnification*

Many of the large type sizes can be obtained satisfactorily by magnifying ten-point type to larger sizes just as *troff* does. Indeed, such magnified fonts may be the only available fonts in the desired size. One may specify a magnified font by using the word **scaled** in the font definition as shown:

```
\font\twelverm=amr10 scaled 1200
```

The scaling factor ("1200" in this example) is based on 1000 as a 1:1 ratio; in this example, it specifies that the normal ten-point letters will be magnified by a factor of 1.2000 to a size of twelve points.

Plain TEX provides a series of standard magnification steps that give a series of magnifications of 1.2 times. They are specified by a macro called `\magstep` as follows:

specification	*value*	*Changes 10pt to*
`\magstep 0`	1000	10pt (no change)
`\magstephalf`	1095	11pt
`\magstep 1`	1200	12pt
`\magstep 2`	1440	14pt (a little more)
`\magstep 3`	1728	18pt (a little less)
`\magstep 4`	2074	21pt (a little less)
`\magstep 5`	2488	24pt (a little more)

Thus, a specification like

```
\font\elevenrm=amr10 scaled \magstephalf
```

produces a reasonable eleven-point roman font.

Finally, one can use magnifications to enlarge the entire contents of a paper (if all the fonts are available in the appropriate magnifications on your installation) by saying `\magnification=` *n* where *n* is an integer such as 1000 (for no change) or can be a `\magstep` macro. Thus, if you say `\magnification=\magstephalf` at the beginning of your paper, its entire contents will be scaled up to eleven-point (well ... 10.95-point) type. Consult your local TEX guru for a list of the fonts and sizes available on your facility.

3.3. *Ligatures*

As with *troff*, many TEX fonts provide ligatures that automatically occur when certain consecutive characters are read. For example, the letters `ff` in the input will produce the single glyph 'ff' on the page. This is intended to improve the appearance of sentences such as "If you flex your fingers in a coffin, you can baffle a giraffe." In TEX, as with *troff*, this happens automatically, and the writer rarely needs to think about it. However, in TEX, there are additional special ligatures that deserve particular attention. Two consecutive hyphens (--) produce an en-dash ('–'; used for ranges such as 'pages 13–34'); Three consecutive hyphens (---) produce a regular punctuation dash (—). Two consecutive closing single quote characters (`''`) will produce a closing double-quote (") while two consecutive opening single-quotes (``` `` ``` —often referred to on terminal keyboards as 'grave accent' characters) will produce an opening double-quote (`"`). The double-quote character " should not be used. Finally, two "Spanish" ligatures are available: the inverted question mark '¿' can be typed as a question mark followed by an open quotation mark(?`), and the inverted exclamation point '¡' can be typed as !`. These are *not* control sequences, but are a property built into the definition of each font. Thus, they may not occur in every available font (e.g. typewriter font), although all the italic, bold, slanted and roman fonts can be expected to behave this way.

4. Composing Text

4.1. *Paragraphs and All of That*

99% of the time, *troff* is reading text (with the occasional font change), filling up output lines as best it can, and outputting them (with occasional interrupts at page breaks).

TEX is doing something similar 99% of the time. It reads text (with occasional font changes) into a buffer. Whenever an end of paragraph is found (the `\par` macro or an empty line), TEX breaks the text in its buffer up into lines, breaking those lines at the points which will produce the most attractive output. For example, it may space out an early line a little more to keep a line with a big word in it from looking too spaced out later. Because TEX composes paragraphs, not lines, its output can look better, and strange tricks are possible. For example, the last sentence of this paragraph ends flush with the right margin because I told TEX to allow no space at the end of the paragraph. Another result of setting paragraphs is that TEX can usually avoid "widow" lines at the top and bottom of pages, setting paragraphs a little more spaced out, if necessary.

One useful feature of the line-breaking algorithm for TEX is found in the use of *ties*. Ties are typed using the character `~` (the tilde character), and behave just like spaces, except that TEX will never split a line at a tie. In *troff*, the special sequence '`\␣`' (backslash-space) is used similarly, but in *troff* this is also an "unpaddable" space that cannot be stretched; in TEX, the only effect of the analogous tie character, the tilde, is to prevent line breaking. There are no strict rules for the placement of ties; they should be used according to stylistic considerations, in places where a line break is unacceptable. For example, a phrase like 'Chapter 2' should be typed as '`Chapter~2`' to avoid something that looks like 'Chapter 2'. Other good places for ties include (1) after the number in enumerated cases like these; (2) The space between the surnames in phrases like 'Vincent Van Gogh'; (3) Between math symbols in apposition as in 'dimension d'.

Frequently, one wishes to change the normal paragraph indentation, or indent a paragraph at both margins, or adjust the amount of space between paragraphs. *troff* allows one to set registers and specify local motions for these operations. TEX allows something superficially similar, but the actual mechanism is quite different. A discussion of *glue* is thus appropriate.

4.2. *Boxes and Glue*

Because *troff* targets its output for a phototypesetter that produces its output on a continuous form, the composition of elements on the page involves commands to move up and down the page, or left and right. Even a 'begin page' command just causes movement downward to the length of the current page[8].

TEX, however, composes pages as units rather than as a stream of commands, and builds them from blocks of text called *boxes* (typically paragraphs), much as hand-set print was built from composing sticks filled with letters. The blank space around the boxes (such as the paragraph indentation or the space between lines) is called *glue*—it is much like the wedges, or *quoins*, that printers used to set a page of print. Thus, while a *troff* command like `.sp .4i` means "move down .4 inches," the corresponding TEX control sequence, `\vskip .4in` means "at this point, there should be .4 inches of glue." Although the difference may seem to be minor, TEX's behavior at paragraphs, page breaks, and so on can only be properly understood if the picture of "boxes of text separated by glue" is kept in mind. Of course, these boxes should not be confused with the boxes that can be drawn around "boxed tables" in *tbl*, or the `.bx` macro in the `-me` package. They are simply the units into which text material is organized on the page.

Of course, the boxes themselves are often composed of other boxes, surrounded by glue. The page that gets output is a box made up of paragraph boxes separated by vertical glue; these paragraph boxes are composed of line boxes with vertical glue (baseline spacing) between them; and these line boxes are in their turn made up of word boxes with horizontal glue (interword spacing) between them. If one wants to get

[8] This causes problems when the phototypesetter is simulated on a page-oriented device and a page length other than 11 inches is specified.

really fancy, one can even put together a box explicitly with \hbox (for horizontal boxes) and \vbox (for vertical boxes), as in

```
for \vbox{\hbox{'A box}\hbox{of Text.'}}
```

'A box
for of Text.' This is a vertical box (\vbox) composed of two horizontal boxes (\hbox)—the first contains "'A box", the second "of Text.'". Such constructions are usually undertaken by TEXperts, and are rarely needed in ordinary applications. However, one special type of box, called a *rule*, is frequently useful. A rule is a box that is filled entirely with ink. Normally, rules are very thin and are used for horizontal and vertical lines. The control sequences \hrule and \vrule are used to construct horizontal and vertical rules respectively. For example, to construct the little blot '▮', I typed

```
the little blot '\vrule height 8pt width 4pt'.
```

Vertical rules should be used within paragraphs, horizontal rules between paragraphs, even if this means having to specify a wide, flat "vertical" rule in order to get a horizontal line into a paragraph. Rules have default thicknesses of 0.4pt, appropriate to thin lines.

Finally, a rule will have a default length equal to the box which contains it. This can sometimes be confusing, but one important result is that a horizontal rule, when between paragraphs, is as wide as the page. This can be useful in setting off diagrams and the like. For example, \hrule was typed after this paragraph.

4.2.1. *Dimensions*

Just as *troff* lets you describe distances in inches, points, ems, etc., TEX permits a variety of units in which the dimensions of glue and boxes can be specified:

pt	point (baselines in this paper are 12 pt apart)
pc	pica (1 pc = 12 pt)
in	inch (1 in = 72.27 pt)
bp	big point (72 bp = 1 in)
cm	centimeter (2.54 cm = 1 in)
mm	millimeter (10 mm = 1 cm)
dd	didôt point (1157 dd = 1238 pt)
cc	cicero (1 cc = 12 dd)
sp	scaled point (65536 sp = 1 pt)

In addition, there are two other context-dependent units of measure whose actual size depends of the current font:

em is the width of a "printer's em" in the current font.
ex is the "x-height" of the current font.

The "em" will be familiar to *troff* users—it is the same as the m unit used in such *troff* parameters as 10m, and, in the days of mechanical type, was based on the width of the letter 'M'. This is typically the same as the point size of the current font. Similarly, the "x-height" is nominally the height of the letter 'x', although this may not be exactly true in every font.

In fact, each font defines its own em and ex values. In 10-point roman, the font you are now reading, 1 em = 10 pt and 1 ex ≈ 4.3 pt. The figures for **Bold face** type of the same size are 11.5 and 4.44 respectively.

4.2.2. *Using Glue*

When glue is placed into the document, it appears as white space. TEX provides several control sequences that place glue into the document. \hskip 1in will insert one inch of horizontal glue (i.e. a one-inch space) between words at the current position in the document. \vskip 8pt will terminate the current paragraph

and follow it with 8 points of (vertical) glue. One very useful type of horizontal glue is **\quad**, which simulates a "printer's quad", i.e. a 1-em space.

There are also three types of useful vertical glue defined as plain TEX macros, which are sometimes helpful in formatting. The control sequence **\smallskip** generates a 3-point skip, **\medskip** a 6-point skip, and **\bigskip** a 12-point skip.

This line is followed by a **\smallskip**,

this by a **\medskip**,

and this by a **\bigskip**.

The sizes of these skips are independent of the current font size.

Several parameters can be set to govern the glue to be placed at critical points in the document. For example, the glue between lines in this document is set so that the distance between baselines is 12 points. This can be adjusted by saying **\baselineskip=14pt**, for example, if wider spacing is desired (e.g. for 12-point type). Here are some other useful parameters:

Name	*Default*	*Description*
\parskip	*0pt*	Space between paragraphs
\leftskip	*0pt*	Placed at left of each line of a paragraph
\rightskip	*0pt*	Placed at right of each line of a paragraph
\topskip	*10pt*	Glue at top of page
\parindent	*20pt*	Paragraph indentation
\hsize	*6.5in*	Width of text on the page
\vsize	*8.9in*	Vertical size of text area on the page

4.2.3. *Stretching and Shrinking*

Actually, glue is more than just rigid wedges between boxes. It also has a certain amount of elasticity. For example, when TEX sets a line of text, the glue between the words in the line is able to shrink or stretch a certain amount in order to adjust that line's spacing. Similarly, the **\parskip** glue between paragraphs is actually able to stretch an additional point if necessary to improve the appearance of a page, and the **\smallskip** series of skips can shrink or stretch by as much as 33% of their natural lengths to assist in page formatting.

Some glue is infinitely elastic. If you think of the printer's page filled with rigid steel boxes of type and wedges between, this infinitely stretchable glue is like a powerful spring. The name of horizontally stretchable glue is **\hfil**[9]. If you type **\line{\hfil Text}** then TEX will set a line of type the width of the page, with the word "Text" at the far right end. The **\centerline** macro seen at the beginning of this paper is a **\line** with springs at *both* sides of the text.

4.3. *Up, Down, Left, Right*

We've already seen how to add vertical and horizontal glue (effictively skipping down and right) to create "white space" on the page. One can also use vertical and horizontal glue with a *negative* dimension to move upwards or to the left, although this is rarely useful. There are a few other useful primitives for "moving around" the page. The primitive control word **\kern** specifies unstretchable glue. The name derives from a printer's term referring to type that projects beyond the body of a letter. **\kern** is normally used

[9] You can also specify such glue as **\hskip** or **\vskip** glue, with a dimension of **0pt plus 1fil**. The **plus 1fil** tells TEX that this glue can stretch to "one degree of infinity" as needed. You can even make it "infinitely shrinkable" by saying **0pt minus 1fil**, or both at the same time.

between letters to move them further apart or closer together (`\hskip`, nearly identical in semantics, can be used as well). For example, the letters 'T' and 'E' in "TEX" have been moved closer together by the command `\kern-.1667em`. Notice the size of this glue is negative and pulls the letter boxes on either side closer together than normal. The 'E' in 'TEX' has also been lowered; this was done with the control sequence `\lower .5ex\hbox{E}`. This is rather complex. The `\hbox{E}` places the letter 'E' in its own horizontal box; the `\lower .5ex` sequence moves the baseline of that box lower by .5ex. Similarly, one can raise a box above its normal baseline by using `\raise` instead of `\lower` (or by using `\lower` with a negative dimension). Normally, someone setting a simple document will not need to set up his or her own \hboxes, just as one rarely requires the horizontal and vertical movements of *troff*.

Sometimes, one wishes to set type someplace and have it set as if it takes up no space at all (e.g. for overstriking). Plain TEX provides two macros, `\rlap` and `\llap` for this. `\rlap{something}` sets 'something' and then backspaces as if nothing had been set at all. Similarly, `\llap{something}` sets 'something' at the left of the current point, overlapping whatever was on the left. Using typewriter font, for example, one can typeset '≠' by saying '`\rlap{/}=`' or '`/\llap{=}`'.

5. Page Composition

5.1. *Headers and Footers*

Plain TEX normally produces pages with no header lines and a page number centered at the bottom. If something else is wanted, macros are provided to change them. `\footline` defines the footer line. For example, a paper might set up a footer line with the command:

```
\footline={Rogue for Pacifists\hfil\folio}
```

to produce a footer line like:

Rogue for Pacifists 9

The alert reader will have noted the use of `\folio` for the page number. This useful (if somewhat cryptically named) macro produces the Arabic representation of the page number if it is positive, and the Roman numeral representation of the number if it is negative. The page number begins at 1 and is incremented after every page (or decremented if it is negative). If you want to adjust it yourself, you can issue the command `\pageno=`*n*, where *n* is the (possibly negative) new page number.

The header line can be set up the same way, using `\headline=`. In fact, the document *Using Nroff and -ME*, by Eric Allman, uses a header line exactly analogous to the footer line of the previous example.

Three-part headers and footers are trickier, because just putting `\hfil` glue between the parts will center the middle section between the left and right sections, rather than centering it on the page. The control sequences `\rlap` and `\llap` can be used to good effect here, since their results behave as if they are of zero width. Think of this as "allowing the `\hfil` springs to push against the sides of the page," and you get the idea. For example, the footer line of this page was set with the input

```
\footline= {
\tenrm                  % Ten-point Roman
\rlap{\TeX\ For {\it Troff\/} Users} % Takes no space, types to right
\hfil\folio\hfil        % Centered Page number
\llap{Sample}           % Also no space, types to left
}
```

♠ A K 8 4
♡ K 3 2
◇ 5
♣ A K J 9 6

Figure 1. Example of a figure

5.2. *Footnotes and Figures*

Most documents require some kind of "inovable" text, i.e. text that is to be set somewhere other than where it appears in the input. In general, TEX calls such items *inserts* and provides powerful, if complex, primitives for the effective manipulation of such material. Fortunately, plain TEX provides fairly handy macros for doing some of the more familiar types of inserts.

The most common insert is a footnote, which consists of text to go at the bottom of a page. The plain TEX `\footnote` macro can be used within a paragraph*; for example, the footnote in the present sentence was typed in the following way:

```
... within a paragraph\footnote{*}{Like this.}; for example, ...
```

There are two parameters to `\footnote`: the reference mark (which will appear in the footnote and in the paragraph), and the text of the footnote. If necessary, long footnotes are automatically split between pages, and footnotes normally appear in normal sized type as in this example**. Note that footnotes *must* appear in the context of a paragraph. If they occur between paragraphs, or in a `\centerline`, or whatever, they will be lost; the esoteric macro `\vfootnote` should be used under these circumstances.

The second kind of insert that many people use is exemplified by the "floating keep" construct of the `-me` macro package. In this case, an illustration or figure is to be kept together and set onto the page at the earliest convenient time. There are three kinds of floating inserts in plain TEX. The first, `\topinsert`, places its material, which ends with `\endinsert`, at the top of the current page, if possible, otherwise the material will be placed at the top of the next page. For example:

```
\topinsert
\centerline{\vbox{
\hbox{$\spadesuit$ A K 8 4}
\hbox{$\heartsuit$ K 3 2}
\hbox{$\diamondsuit$ 5}
\hbox{$\clubsuit$ A K J 9 6}
}}
\vskip 12pt
\centerline{{\bf Figure 1.} Example of a figure}
\endinsert
```

`\topinsert` automatically adds some extra space to separate the caption from the following text.

The second floating insert, `\midinsert` is much more like the `.(z` construct of the `-me` package; it tries to insert the material in place in the middle of the current page. If it succeeds, the material is placed in a vertical box with about 12 points of space above and below. If this can't be done, the effect is that of a `\topinsert`.

Finally, the macro `\pageinsert` justifies its material to the size of a full page and places the result on

---

* Like this.

** This paper uses a private macro (not part of plain TEX) to automatically number footnotes and display them in eight-point type.

the following page. For example, the following produces a familiar result:

```
\pageinsert
\vfil % vertical centering
\centerline{This page intentionally left blank}
\vfil % vertical centering
\endinsert
```

5.3. *Other Keeps*

Plain TEX doesn't provide all the macros that one might want to use for keeps, displays, and the like. However, it's fairly easy to tell TEX to try to avoid breaking up a group. Simply set the cryptic parameter `\interlinepenalty=150` within that group, and this will make line breaks within the group into very poor candidates for page breaking. Another strategy is to use the control word `\filbreak`. `\filbreak` causes the remainder of the current page to be filled with vertical glue, **unless** it can be filled with additional text that is itself followed by `\filbreak`. If every paragraph had `\filbreak` after it, every page of a document would have only complete paragraphs on the page.

Similarly, while there is no specific "display" macro to correspond to the `.(l` macro of **-me** or the `.DS` macro of **-mm**, one can specify `\obeylines` within a group to cause each newline to begin a new paragraph. One might also wish to set `\parindent` to an appropriate value for the lines within the group. You can even have an entire group of centered lines by specifying `\obeylines`, setting `\parindent` to zero, and `\leftskip` and `\rightskip` (i.e. the left and right margins) to be infinitely extensible glue[10]. There may be some prepackaged macro facilities available on your system for doing these sorts of things.

6. Tables

Users of *troff* often find themselves setting up tables in their documents by using *tbl*, a program that turns (relatively) simple table descriptions into the complicated commands that unvarnished *troff* requires to set the document. Plain TEX has its own built-in facility for the creation of small (less than one page) tables. While TEX table descriptions may be somewhat harder to read than the corresponding input for *tbl*, the table mechanism is well-integrated with the other TEX facilities, and there is no preprocessor required.

6.1. *Tabs*

The easiest kind of table is one that uses "tab stops" set regularly across the page, analogous to the physical stops in a typewriter. In TEX, one can set these tab stops by saying `\settabs` *n* `\columns`. This divides the line into *n* equal parts. To use the tab stops, one types text that begins with the control sequence `\+` and ends with the sequence `\cr`. "`\cr`" stands for Carriage-Return, and suggests a time when typewriters had carriages. If you forget to end a tabbed line with `\cr`, TEX will become very confused.

The TEX tab character is the ampersand, **&**; while this can be changed to something else, the syntax-changing commands of TEX are (deliberately) somewhat opaque, and you probably don't want to try it. Here's an example of using tabs (stolen shamelessly from "*The TEXbook*"):

```
\settabs 4 \columns
\+&&Text that starts in the third column\cr
\+&Text that starts in the second column\cr
\+\it Text that starts in the first column, and&&&
the fourth, and&beyond!\cr
```

The result of all this is:

[10] i.e. `\leftskip=0pt plus 1fil` and `\rightskip=0pt plus 1fil`

This page intentionally left blank

	Text that starts in the second column	Text that starts in the third column	
Text that starts in the first column, and		the fourth, and	beyond!

There are several interesting features in this example. The ampersand isn't exactly like a mechanical tab because it backs up, if necessary, to reach the next numerical tab stop; this makes tab-based tables somewhat easier than their *troff* counterparts, in which one must always know when a field will slop over into the next tabbed column. Thus, in the last line, three &'s were required to get to column 4, even though the first entry had extended into column 2. Lines 2 and 3 show that the `\cr` can end a line even if some fields are not specified. The last pair of lines shows that spaces are ignored after ampersands; hence you can end an input line harmlessly at such a point without getting extraneous spaces. The last line also shows that each individual entry in a tabbed line is implicitly grouped as if it were in braces; for this reason, no braces were required around the `\it` section.

The dashed columns, by the way, are pedagogical, and do not normally appear.

6.2. *General Tab Stops*

For somewhat more complicated tables, there's a second form of the **`\settabs`** control sequence, in which a sample line (usually consisting of the widest entries to be found in each field) is supplied in order to set the tab stops. The tabs are placed at the positions of the &'s in the sample line, but the sample line itself will **not** actually appear in the output. There should be some extra space between columns, to prevent the text in adjacent columns from touching. For example:

```
\settabs\+\hskip 1in&Section Three\quad&\cr % Sample line
\+&Section One&Commands\cr
\+&Section Two&System Calls\cr
\+&Section Three&Subroutines\cr
```

causes the following three lines to be typeset:

Section One	Commands
Section Two	System Calls
Section Three	Subroutines

Notice that the sample line will typically end with **`&\cr`**, because the text following the last tab isn't used for anything. In other words, the **only** thing a sample field does is determine the amount of space to the *next* tab stop. Also notice that each tabbed line in this example begins with a tab, so as to get the 1-inch indentation (**`\hskip 1in`**) that was specified for the first field width in the sample line.

Tabbed entries can also contain stretchable glue[11], allowing one to do centering and justifying. For example:

```
\settabs 2 \columns
\+\hfill This material&\hfill While this\hfill&\cr
\+\hfill is flush right&\hfill material is centered\hfill&\cr
\+\hfill in the left-hand column&\hfill in the right-hand column.\hfill&\cr
```

produces the following:

This material	While this
is flush right	material is centered
in the left-hand column	in the right-hand column.

[11] **`\hfill`** glue (with two 'l's) must be used here, as you need something "infinitely more stretchable" than the normal **`\hfil`** glue that TEX uses to left-justify the tabbed entries. In effect, the **`\hfill`** glue overrides the normal glue.

6.3. *Alignments*

Finally, there is a general mechanism, known as *alignment*, which can be used to set generalized tables. The details of alignments can get quite complex, and a full chapter is devoted to them in "*The TEXbook.*" However, a short explanation and example may be of interest. The command **\halign** is used to set up arbitrary tables. The general format is

\halign{ *Line Template* **\cr**
table lines, each ending with **\cr**
}

Each table line will be "plugged into" the line template, with the individual line elements (separated, as before, with ampersands) substituted wherever a pound-sign (#, also known as 'sharp-sign' or 'octothorpe') occurs in the template. As a specific example, suppose we want to set the following table:

Section	*Subject*
1	Commands
2	System Calls
3	Subroutines
4	Special Files
5	File Formats
6	Games
7	Miscellany
8	System Maintenance

To accomplish this, the line template must include the **\hfil** glue used to center and justify the columns, and the boldface shift for the numerals that appear in the first column. In fact, it looks like this:

```
\hfil\bf#\hfil & \quad#\hfil\cr
```

The first column,' **\hfil\bf#\hfil**' will automatically have its text, represented by the pound-sign, centered by the two **\hfil** commands, and emboldened by the **\bf**. As with tabbing (which is really just a special case of alignment), the font change is implicitly grouped within the single table entry. The second column, '**\quad#\hfil**' will have a 1-em padding at the beginning, and will be left-justified by the **\hfil** glue at the end. Here's the whole command:

```
\halign
{\hfil\bf#\hfil & \quad#\hfil\cr
\sl Section&\sl Subject\cr
&\cr
1&Commands\cr
2&System Calls\cr
3&Subroutines\cr
4&Special Files\cr
5&File Formats\cr
6&Games\cr
7&Miscellany\cr
8&System Maintenance\cr}
```

Notice how the normal fonts were overridden in the title line, and how an empty line was set to separate the title line from the remainder of the table. As an alternative way to get some space between lines in an alignment, you can say

\noalign{ *material* **}**

after a `\cr` in an alignment and that material will be copied in place without being aligned. For example, '`\noalign{\smallskip}`' is an easy way to get a small skip between two lines of a table.

There was also some other magic involved in centering the table on the page, to be described in the next section. Basically, the alignment was placed in its own `\vbox`, and that vertical box was horizontally centered.

The examples of this section have been simple ones, but cover most of the types of tables that are used in most documents (such as the tables of dimensions or special characters used in the earlier sections of this document). People with more complex needs will have to consult a TEX guru, or learn more from the chapter on alignments in "*The TEXbook*."

7. Setting Mathematics

Many people find that the most useful adjunct to *troff* is the program *eqn*, which provides a "pronouncable" way to describe mathematical equations for typesetting. TEX provides a built-in *mathematics mode* in which the meanings of the letters and control sequences are changed to allow easier equation settings.

Actually, there are two forms of mathematics mode in TEX: *text* math mode, and *display* math mode. Text math mode is used for the in-line expressions specified by the `delim` feature of *eqn*. It is delimited by single dollar signs. For example, one can say something like

```
Let $x$ be the sum of $y$ and $z/2$.
```

in the middle of a paragraph, and produce something like 'Let x be the sum of y and $z/2$.' Note that math mode sets letters in italics but numerals in roman font.

Display math mode is for setting equations between paragraphs just as the `.EQ` and `.EN` macros of *eqn* do. The display math delimiters are double dollar signs, `$$`. For example, to display the equation

$$x + 5$$

this document includes a line that says:

```
to display the equation $$x+5$$ this document includes
```

Note that display math mode automatically centers the display on the page and provides the extra vertical spacing around it.

TEX's math mode contains a great deal of built-in knowledge about equation spacing, and therefore *ignores* spaces that occur between the dollar signs (just as *eqn* does). For example `$ x $` has the same effect as `$x$`. You can always force extra space by using `\quad`, `\␣` (escape-space), or a glue specification; there are also special control sequences for thinner spaces.

7.1. *Greek Letters*

Unlike *eqn*, which converts words like 'omega' into the corresponding Greek letters, TEX requires control words to produce these. While this may, at first, seem a nuisance, one is freed from having to worry about 'reserved words' in TEX. For example, to produce

$$x = 2\pi \int \sin(\omega t) dt$$

one would type `$$x=2\pi\int\sin(\omega t)dt$$`. Upper-case Greek letters (which are pronounced like `GAMMA` in *eqn*) have names like `\Gamma` in TEX. Math mode provides a host of other math characters like `\infty` for '∞'; the reference card provided with this document lists several of these.

7.2. *Subscripts and Superscripts*

TEX math mode provides a simple method of superscripting and subscripting using the special characters ^ and _, respectively. For example, to set 'x^2' one types `$x^2$`. Similarly, 'α_0' is pronounced `$\alpha_0$`. These characters normally apply only to the next single character. If you want more things subscripted or superscripted, you can group them. For example, 'x^{y_2}' is typed as `$x^{y_2}$`. However, unlike *eqn*, TEX considers a construct like `$x^y^z$` illegal; you should specify `$x^{y^z}$` or `$x^{yz}$`, depending on what is meant.

One can, however, specify `$x^2_3$` in order to obtain 'x^2_3'; `$x_3^2$` is equivalent. Notice that simultaneous $\mathrm{su}_{\mathrm{b}}^{\mathrm{per}}$scripts are stacked over one another. A special character '$\prime$', designed specifically for being shrunk and raised for superscripting, is designated by the control word `\prime`. For example, one might refer to `$f^\prime(x)$` to set '$f'(x)$'. Plain TEX provides a convenient abbreviation: a single quote. For example, you could also say `$f'(x)$` (for '$f'(x)$') or even `$f'''(x)$` for '$f'''(x)$'.

7.3. *Fractions*

TEX fractions are similar to *eqn*'s treatment. One can type `$a/b$` to obtain 'a/b,' but if one wants an equation like

$$\frac{x+y^2}{k+1}$$

one may type '`$${x+y^2}\over{k+1}$$`'. This can be taken to excess; Knuth warns one against expressions like `$$a\over{b\over 2}$$`, which produces the formula

$$\frac{a}{\frac{b}{2}}$$

This looks fairly awful; the recommended alternative is `$$a\over{b/2}$$`, which produces

$$\frac{a}{b/2}$$

Plain TEX also provides an operator `\choose` for producing binomial coefficients such as $\binom{n}{2}$, which is typed as `$n\choose 2$`.

7.4. *Square Roots*

Square roots are obtained via the control sequence `\sqrt`. For example `$\sqrt2$` produces '$\sqrt2$' and `$$\sqrt{x^3+\sqrt\alpha}$$` produces

$$\sqrt{x^3+\sqrt{\alpha}}$$

TEX is able to handle fairly tall formulas without getting too ugly. For example, the input

```
$$\sqrt{a^2\over{b_2}}$$
```

produces

$$\sqrt{\frac{a^2}{b_2}}$$

which is substantially better than the ugly example on page 4 of the *eqn* manual. A similar method produces lines below or above formulas: `$\overline{x+y}$` produces '$\overline{x+y}$.'

7.5. *Large Operators*

Plain TEX provides *large operators* like $\sum$, $\int$ and $\prod$, which produce larger symbols in display math mode than in text. For example `$\sum x_n$` produces $\sum x_n$, but `$$\sum x_n$$` produces

$$\sum x_n$$

If one wishes to add "limits" to such operators, they can be typed like subscripts. For example, `$$\sum_{n=1}^m x_n$$` produces

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

The `\int` operator, $\int$, normally has its limits placed to the sides of the operator. For example `$$\int _0^{\infty}$$` produces

$$\int_0^{\infty}$$

If one wishes to change this convention, one can type '`\limits`' directly after the `\int` operator. For example, `$$\int\limits_0^{\pi\over 2}$$` yields

$$\int\limits_0^{\frac{\pi}{2}}$$

Certain defined control sequences in plain TEX also accept limits. For example,

$$\lim_{n\to\infty} x_n = 0$$

is produced by `$$\lim_{n\to\infty}x_n=0$$`.

7.6. *Fonts*

Normally, text in math mode is set in math italic (nearly identical to text italic). Sometimes, one wants roman type as part of a formula, especially with such mathematical functions as 'log' and 'sin'. Plain TEX defines several control sequences such as `\sin`, `\ln`, and `\lim` which always are set in roman type. You can also switch explicitly to roman by typing `\rm`. For example, 'x^3 + lower order terms' can be set by typing `$x^3+{\rm lower\ order\ terms}$`. Notice that spaces had to be explicitly inserted by preceding them with the backslash, because TEX ignores spaces in math mode.

Bold face can also be used. For example, `$\bf a+b=\Phi_m$` produces '$\mathbf{a+b=\Phi_m}$'. Plain TEX arranges matters so that the `\bf` control sequence only affects alphabetic characters when in mathematics mode. There is also a 'calligraphic' font for use with upper-case (and *only* upper-case) letters in math mode. `$\cal EXAMPLE$` produces '$\mathcal{EXAMPLE}$'. Finally, `\it`, `\sl`, and `\tt` can be used, but cannot be produced in subscript size.

7.7. *Accents*

Plain TEX defines eight control sequences for placing accents over letters in mathematics mode:

`$\hat a$`	$\hat{a}$
`$\check a$`	$\check{a}$
`$\tilde a$`	$\tilde{a}$
`$\dot a$`	$\dot{a}$
`$\ddot a$`	$\ddot{a}$
`$\breve a$`	$\breve{a}$
`$\bar a$`	$\bar{a}$
`$\vec a$`	$\vec{a}$

Note that the **dyad** accent of *eqn* is not present in plain TEX. Also note that **\underline** and **\overline** can be used to place a bar over or under any formula.

7.8. *Alignments*

The **mark** and **lineup** constructs of *eqn* are used to align equations. In TEX, this can be done using the powerful **\halign** mechanism as described in the previous section. However, plain TEX also provides some special-purpose alignment macros for doing some of the more common operations without resorting to **\halign**.

The most common use of aligned formulas is in multi-line displays that should be lined up by their '=' signs. Plain TEX provides the **\eqalign** macro for this purpose. For example, the example (that can't be done with **mark**) in the *eqn* manual:

$$x = 1$$
$$x + y = z$$

was typed as

```
$$\eqalign {
x &=1 \cr
x+y &= z\cr
}$$
```

The right-hand side can start with an equals-sign or any other symbol. For example, one might wish to say something like:

$$x + y + z < 5$$
$$z = y/42$$
$$y \sin z > x \log z$$

with the equality and inequality symbols aligned. This can be typed as

```
$$\eqalign {
x+y+z &< 5\cr
z &= y/42\cr
y\sin z &> x\log z\cr
}$$
```

Another type of aligned display is something like

$$f(x) = \begin{cases} x, & \text{for } x \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

For this sort of display, the special macro **\cases** is defined. This example was typed as

```
$$f(x)=\cases{
x,& for $x\ge 0$;\cr
0,& otherwise.\cr}$$
```

Note that the first column is implicitly in mathematics mode, but the second column is *not*. For example, the '$x \geq 0$' of the first line had to be set explicitly in math mode. The **\cases** macro automatically typesets its own '{' in the appropriate size.

Finally, matrices can be set up using the **\matrix** alignment macro. For example, a neat array like

$$\begin{matrix} x_i & x^2 \\ y_i & y^2 \end{matrix}$$

can be set up by typing

```
$$\matrix{
x_i & x^2 \cr
y_i & y^2 \cr
}$$
```

To set the array with big parentheses around it, `\pmatrix` can be used instead of `\matrix`, to produce

$$\begin{pmatrix} x_i & x^2 \\ y_i & y^2 \end{pmatrix}$$

Of course, there are other ways of typing arbitrary brackets, as described in the next section.

7.9. *Big Brackets, Etc.*

Plain TEX provides an assortment of brackets and delimiters that can be used for formulas. The obvious ones are the parentheses and brackets; one can also use `\{` and `\}` for braces in math mode. In addition, there are `$\lfloor$` for '⌊', `$\lceil$` for '⌈', and `$\langle$` for '⟨'. The right-hand versions are `$\rfloor$`, `$\rceil$`, and `$\rangle$`. One can also use the vertical bar, |, as a delimeter (e.g. to signify absolute value).

Sometimes one wishes to get a larger version of these symbols. To do this, precede them by '`\bigl`' for the left side and '`\bigr`' for the right. This can make formulas easier to read. For example,

```
$$\bigl(x-s(x)\bigr)\bigl(y-s(y)\bigr)$$
```

produces

$$\bigl(x - s(x)\bigr)\bigl(y - s(y)\bigr)$$

In *eqn*, there is a generalized mechanism that produces brackets big enough for whatever they enclose, using the keywords `left` and `right`. The mechanism in TEX is quite similar. For example, the example on page 6 of the *eqn* manual,

$$\left\{ \frac{a}{b+1} \right\} = \left(\frac{c}{d} \right) + [e]$$

is produced by the input

```
$$\left\{a\over b + 1\right\}
=\left(c\over d \right)
+\left[e\right]
$$
```

The `\left` and `\right` delimiters *must* pair up with one another, just as braces do in TEX groups. However, one can use a period ('.') as a null delimiter in unbalanced groups. For example, `$$x=\left\{y\over 3\right.$$` produces

$$x = \left\{ \frac{y}{3} \right.$$

The '`\right.`' sequence outputs nothing, but closes the group begun by the `\left\{` sequence.

7.10. *Displayed Text*

Display math mode can also be used to center and set off textual material or alignments. To do this, simply place the material in a box using `\hbox` or `\vbox`. For example,

```
$$\hbox{Display Text}$$
```

produces

$$\hbox{Display Text}$$

Notice that the text occurs in the regular text font (ten-point Roman) rather than math italics. An even more practical use of this technique is for alignments. A table can be centered easily.

```
$$\vbox{\halign {\tt#\hfil &\quad #\hfil\cr
\bf\hfil Name & \bf\hfil Supplier\cr
\noalign{\smallskip\hrule\smallskip}
\TeX & University of Washington\cr
TROFF & Bell Laboratories\cr
SCRIBE & Unilogic, Ltd.\cr
DSR & DEC (VMS Only)\cr
}}$$
```

produces

$$\vbox{\halign {\tt#\hfil &\quad #\hfil\cr
\bf\hfil Name & \bf\hfil Supplier\cr
\noalign{\smallskip\hrule\smallskip}
\TeX & University of Washington\cr
TROFF & Bell Laboratories\cr
SCRIBE & Unilogic, Ltd.\cr
DSR & DEC (VMS Only)\cr
}}$$

complete with the spacing that separates it from the surrounding textual material.

7.11. *More Reading*

This section has outlined some of the more straightforward uses of math mode. There are many other facilities, as described in "*The TEXbook*". Consult that volume for additional ideas.

8. Macros and Definitions

We have already shown one instance in which a TEX control sequence was defined or modified by the user. This was the use of `\font` to define a font-switching control sequence. However, one can also define a new TEX control sequence by using the primitive `\def`. Such new control sequences are called *macros* as they are, in effect, short abbreviations for longer streams of commands. It is, of course, possible to define whole families of macros similar in function to the macro packages such as **–me** or **–mm** in *troff*, or the 'document formats' such as **@Make(Report)** of **Scribe**. Knuth's TEXbook was produced using such a package, and the numbered section headings and footnotes for this document were produced by specialized macros such as the `\Section` macro seen in the example at the beginning of the document.

However, even when using only the built-in facilites of plain TEX, which are adequate for many kinds of papers, or even a more complex macro package, one sometimes wishes to define often-used sequences. For example, this paper uses a macro, `\Unix`, which is similar to `\TeX`, but produces 'UNIX' in 10-point small-caps font. It was defined as follows:

```
\font\csc=amcsc10 % Small-caps font
\def\Unix{{\csc Unix}} % Extra grouping for font-change
```

As a result, whenever `\Unix` is seen in the input for this paper, TEX substitutes the sequence `{\csc Unix}`.

If you have a file containing a group of useful macros and definitions, or even just a useful section of text that you wish to repeat, you can use the TEX primitive **\input**, which operates much like the **.so** primitive of *troff*. For example, the definitions of **\Section** and so on for this paper are contained in a file called **paperhead.tex**; the first line of this document is **\input paperhead**. If the file to be input isn't in the current working directory, a "macro library" directory is searched. This allows commonly-used macro packages, such as the American Mathematical Society's $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX package, to be publicly available with a minimum of fuss.

More complex macros with parameters, etc., are possible, but are 'big-league' material outside the scope of this paper. The grouping features of TEX, combined with a powerful set of conditionals, give TEX a structured-language-like appearance, compared to that of *troff*. The Truly Ambitious should read "*The TEXbook*" by Donald Knuth, which is the ultimate font of TEXnical knowledge.

9. Running TEX under UNIX

TEX82 is written in PASCAL and the user interface is nearly identical on the various operating systems to which it has been ported. The TEX source code is placed in a file with a name like '*filename*.**tex**'. The input should end with a **\bye** macro, which automatically fills out the last page with **\vfil** glue, outputs it, and terminates TEX. Assuming a Bourne shell, the command to invoke TEX will be

$ tex82 *filename*

Assuming that **tex82** is found, TEX will print something like

```
This is TeX, Version 0.9999 for Berkeley UNIX (preloaded format=plain 83.8.11)
(filename.tex [1])
Output written to filename.dvi (1 page, 2092 bytes)
Transcript written on filename.log.
$
```

TEX lists the files as it reads them, and displays the numbers of the pages as they are produced (in brackets). It writes error messages and warnings, as well as an assortment of cryptic information, in *filename*.**log** and produces a device-independent ('dvi') representation of the result in *filename*.**dvi**. This file can be sent to your typesetting device by giving it as input to the appropriate device driver program. Consult your local TEXnical guru for information on the device drivers available at your site.

If TEX detects errors in the input, and it has a terminal input device, it will attempt to interact with the user and allow him or her to correct the error. The help facilities built into TEX allow the user to receive an explanation of the error, and provide some guidance as to possible remedies. Type a question-mark and hit the **return** key if you're in doubt. One can, as an alternative, add the command **\batchmode** to the input file to suppress this interaction. In this case, TEX can be run in background mode (redirecting output to **/dev/null**); the output for errors and warnings will still be placed in the appropriate **.log** file for later examination.

10. A Short Example

Pages 22 and 23 contain a short sample of TEX input, demonstrating some of the features discussed in this document. The result is on page 24. Note that the page number in the sample output follows the page numbering in this document, rather than being '1' as it would be if the sample were run separately. The reader should inspect closely the left and right single quotes in the input, which can be rather hard to distinguish in the 'tt' font.

```
\font\titletype=ambx10 scaled \magstep 1 % 12-point boldface
\footline={\TeX\ Sample \hfil Page \folio}
\headline={\hfil{\bf S A M P L E}\hfil}
\vskip .25in
\line{\hfil Sample Output}  % Right-justified
\line{\hfil Feb 30, 1986}   % text due to \hfil
\vskip 12pt
\centerline{\titletype \TeX\ Sample}
\vskip 1cm
This material, which appears in 10-point type (possibly
magnified) , demonstrates
several features:  (1)~Ties.    (2)~Floating Keeps for figures.
(3)~General syntax.  (4)~Equations and tables.    (5)~Verbatim
mode using {\tt obeylines}.

This is a new paragraph.    It contains a footnote\footnote{*}
{Which doesn't tell you much}, and contains some fascinating
accent marks like the accent gr\'ave, and the accents in
phrases like ``\c c'est la vie'' and ``\^gis revido.''    This
paragraph also demonstrated the double-quote ligatures---which
are somewhat hard to read in the 'tt' font---and the long
dashes.    Notice that the interparagraph
spacing is zero, unlike the main body of the paper.
In Figure~1,
we see an equation from ``{\sl A System for Typesetting Mathematics\/}''
by Kernighan and Cherry.
\midinsert % A displayed Equation from Kernighan and Cherry
\def\emx{e^{mx}} % This is done with EQN defines in Kernighan and Cherry
\def\mab{m\sqrt{ab}}
\def\sa{\sqrt a}
\def\sb{\sqrt b}
$${\int\limits{dx\over{a\emx-be^{-mx}}}}=\cases{
{1\over{2\mab}}\log
{{\sa\emx-\sb}\over{\sa\emx+\sb}}&\cr
&\cr
{1\over{\mab}}\tanh^{-i}({\sa\over\sb}\emx)&\cr
&\cr
{-1\over{\mab}}\coth^{-i}({\sa\over\sb}\emx)&\cr
}$$
\centerline{Figure 1.    An Equation}
\endinsert
Furthermore, here is a table from M. Lesk's
``{\sl Tbl---A Program to Format Tables\/}'':
```

```
\vskip 10pt
\settabs\+Language\quad&Carnegie-Mellon\quad&\cr
\+\hfil Language\hfil & \hfil Authors\hfil & \hfil Runs on\hfil\cr %centered
\+ \cr % Skip a line
\+ Fortran & Many & Almost anything\cr
\+ PL/1 & IBM & 360/370\cr
\+ C & BTL & 11/45,H6000,370\cr
\+ BLISS & Carnegie-Mellon & PDP-10,11\cr
\+ IDS & Honeywell & H6000\cr
\+ Pascal & Stanford & 370\cr

\bigskip
Finally, here's an example of {\tt obeylines}:
{\obeylines\parindent=1in\parskip=0pt % Single-space, 1-inch indent

This material
has no interparagraph gap
between the lines, but each
line is, in effect, a new paragraph} % End of Obeylines

Now to finish up.
\bye
```

Sample Output
Feb 30, 1986

TEX Sample

This material, which appears in 10-point type (possibly magnified), demonstrates several features: (1) Ties. (2) Floating Keeps for figures. (3) General syntax. (4) Equations and tables. (5) Verbatim mode using **obeylines**.

This is a new paragraph. It contains a footnote*, and contains some fascinating accent marks like the accent grave, and the accents in phrases like "ç'est la vie" and "ĝis revido." This paragraph also demonstrated the double-quote ligatures—which are somewhat hard to read in the 'tt' font—and the long dashes. Notice that the interparagraph spacing is zero, unlike the main body of the paper. In Figure 1, we see an equation from "*A System for Typesetting Mathematics*" by Kernighan and Cherry.

$$\int \frac{dx}{ae^{mx} - be^{-mx}} = \begin{cases} \frac{1}{2m\sqrt{ab}} \log \frac{\sqrt{a}e^{mx} - \sqrt{b}}{\sqrt{a}e^{mx} + \sqrt{b}} \\ \frac{1}{m\sqrt{ab}} \tanh^{-1}\left(\frac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \\ \frac{-1}{m\sqrt{ab}} \coth^{-1}\left(\frac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \end{cases}$$

Figure 1. An Equation

Furthermore, here is a table from M. Lesk's "*Tbl—A Program to Format Tables*":

Language	Authors	Runs on
Fortran	Many	Almost anything
PL/1	IBM	360/370
C	BTL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

Finally, here's an example of **obeylines**:

This material
has no interparagraph gap
between the lines, but each
line is, in effect, a new paragraph

Now to finish up.

* Which doesn't tell you much

Appendix A
Special characters available in TEX's "math mode"

There are many special-purpose characters available from TEX, but most of them don't appear on most keyboards. Therefore, the following control sequences have been pre-defined in plain.tex for your convenience. Note, however, that these special sequences are *only* available in math mode.

(a) Lower-case Greek letters:

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

(b) Upper-case Greek letters:

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

(c) Script letters (Characters from the \cal *font):*

$\mathcal{A}$	{\cal A}	$\mathcal{J}$	{\cal J}	$\mathcal{S}$	{\cal S}
$\mathcal{B}$	{\cal B}	$\mathcal{K}$	{\cal K}	$\mathcal{T}$	{\cal T}
$\mathcal{C}$	{\cal C}	$\mathcal{L}$	{\cal L}	$\mathcal{U}$	{\cal U}
$\mathcal{D}$	{\cal D}	$\mathcal{M}$	{\cal M}	$\mathcal{V}$	{\cal V}
$\mathcal{E}$	{\cal E}	$\mathcal{N}$	{\cal N}	$\mathcal{W}$	{\cal W}
$\mathcal{F}$	{\cal F}	$\mathcal{O}$	{\cal O}	$\mathcal{X}$	{\cal X}
$\mathcal{G}$	{\cal G}	$\mathcal{P}$	{\cal P}	$\mathcal{Y}$	{\cal Y}
$\mathcal{H}$	{\cal H}	$\mathcal{Q}$	{\cal Q}	$\mathcal{Z}$	{\cal Z}
$\mathcal{I}$	{\cal I}	$\mathcal{R}$	{\cal R}	ℓ	\ell

(d) Binary operators:

$\wedge$	\land (or \wedge)	$\amalg$	\coprod	$\mp$	\mp
$\vee$	\lor (or \vee)	$\perp$	\perp	$\pm$	\pm
$\cap$	\cap	$\diamond$	\diamond	$\backslash$	\backslash
$\cup$	\cup	$\bullet$	\bullet	$\cdot$	\cdot
$\uplus$	\uplus	$\odot$	\odot	$\circ$	\circ
$\sqcap$	\sqcap	$\oslash$	\oslash	$*$	\ast
$\sqcup$	\sqcup	$\otimes$	\otimes	$\star$	\star
$\dashv$	\dashv	$\ominus$	\ominus	$\times$	\times
$\vdash$	\vdash	$\oplus$	\oplus	$\div$	\div
				$\setminus$	\setminus

(e) Binary relations:

$\mid$	\mid	$\notin$	\notin	$\geq$	\geq
$\parallel$	\parallel	$\in$	\in	$\leq$	\leq
$\mid$	\vert	$\gg$	\gg	$\succ$	\succ
$\parallel$	\Vert	$\ll$	\ll	$\succeq$	\succeq
Γ	\bracevert	$\neq$	\neq	$\prec$	\prec
$\mapsto$	\mapstochar	$\sim$	\sim	$\preceq$	\preceq
$\mapsto$	\mapsto	$\simeq$	\simeq	$\supset$	\supset
$\longmapsto$	\longmapsto	$\doteq$	\doteq	$\supseteq$	\supseteq
$\Longleftrightarrow$	\iff	$\cong$	\cong	$\subset$	\subset
$\asymp$	\asymp	$\approx$	\approx	$\subseteq$	\subseteq
		$\equiv$	\equiv	$\sqsubseteq$	\sqsubseteq

$\uparrow$	\uparrow	$\Uparrow$	\Uparrow
$\downarrow$	\downarrow	$\Downarrow$	\Downarrow
$\leftrightarrow$	\leftrightarrow	$\Leftrightarrow$	\Leftrightarrow
$\leftarrow$	\leftarrow	$\Leftarrow$	\Leftarrow
$\rightarrow$	\rightarrow	$\Rightarrow$	\Rightarrow
$\longleftrightarrow$	\longleftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow
$\longleftarrow$	\longleftarrow	$\Longleftarrow$	\Longleftarrow
$\longrightarrow$	\longrightarrow	$\Longrightarrow$	\Longrightarrow
Γ	\arrowvert	Γ	\Arrowvert
$-$	\relbar	$=$	\Relbar

You can also use the control sequence \not to negate or "cross out" most of the relations above. For example, the symbol "$\not\subseteq$" is really two symbols, obtained by typing "\not\subseteq". (The slashing character 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\subseteq}", in order to prevent TEX from breaking a line after \not.

(f) "Large" operators (text and display styles):

$\wedge$	$\bigwedge$	\bigwedge	Σ	$\sum$	\sum	$\cap$	$\bigcap$	\bigcap
$\vee$	$\bigvee$	\bigvee	Π	$\prod$	\prod	$\cup$	$\bigcup$	\bigcup
$\int$	$\int$	\int	$\oplus$	$\bigoplus$	\bigoplus	$\uplus$	$\biguplus$	\biguplus
$\oint$	$\oint$	\oint	$\otimes$	$\bigotimes$	\bigotimes	$\sqcup$	$\bigsqcup$	\bigsqcup
			$\odot$	$\bigodot$	\bigodot			

(g) Brackets:

⌊	\lfloor	⌋	\rfloor
⌈	\lceil	⌉	\rceil
[	\lbrack	]	\rbrack
⟮	\lgroup	⟯	\rgroup
⟮	\lmoustache	⟯	\rmoustache
{	\{	}	\}
⟨	\langle	⟩	\rangle

(h) Miscellaneous math symbols:

				∂	\partial
ı	\imath	∅	\emptyset	′	\prime
ȷ	\jmath	∃	\exists	∫	\smallint
ℜ	\Re	∀	\forall	⊤	\top
ℑ	\Im	△	\triangle	⊥	\bot
ℵ	\aleph	∠	\angle	√	\surd
℘	\wp	¬	\lnot (or \neg)	⨿	\amalg
∞	\infty	∇	\nabla	≀	\wr

(i) Miscellaneous nonmath symbols (math mode required):

♠	\spadesuit	♯	\sharp	⌣	\smile
♡	\heartsuit	♭	\flat	⌢	\frown
◇	\diamondsuit	♮	\natural		
♣	\clubsuit				

(j) Miscellaneous nonmath symbols (math mode not required):

$	\$	#	\#	&	\&
%	\%	§	\S	‡	\ddag
©	\copyright	¶	\P	†	\dag

Appendix B
Font Samples

Plain TEX predefines control sequences for the math italic, bold and roman styles of the Computer Modern family in five-point, seven-point and ten-point (the default). The predefined fonts are as follows:

`\tenrm`	Normal ten-point roman
`\teni`	*ten'pointmathitalic*
`\tenit`	*ten-point text italic*
`\tenbf`	**ten-point bold extended**
`\tensl`	*ten-point slanted roman*
`\tentt`	`ten-point typewriter`
`\sevenrm`	seven-point roman
`\seveni`	*seven'pointmathitalic*
`\sevenbf`	**seven-point bold extended**
`\fivern`	five-point roman
`\fivei`	*five'pointmathitalic*
`\fivebf`	**five-point bold extended**

For other fonts, the `\font` operation must be used to associate a keyword with the font information file. Here are some of the fonts distributed with the UNIX version of TEX:

`amsaq8`	8-point sans-serif
`amss10`	10-point sans-serif
`amssmc10`	**Emboldened 10-point sans-serif**
`amssmc40`	**40-point sans-serif**
`amssqi8`	*8-point slanted sans-serif*
`amssi10`	*10-point slanted sans-serif*
`amssbx10`	**10-point bold sans-serif**
`amr6`	6-point roman
`amr8`	8-point roman
`amr9`	9-point roman
`amrc10`	Narrower 10-point roman
`amsl8`	*8-point slanted*
`amsl9`	*8-point slanted*
`ambxsl10`	***10-point bold slanted***
`ambx6`	6-point bold
`ambx8`	**8-point bold**
`ambx9`	**9-point bold**
`amb10`	**Narrower 10-point bold**
`ambct2`	**Narrower 12-point bold**
`ami6`	6'pointmathitalic
`ami8`	*8'pointmathitalic*
`ami9`	*9'pointmathitalic*

amti7	*7-point text italic*
amti8	*8-point text italic*
amti9	*9-point text italic*
ambi10	***10-point bold math italic***
amtt8	8-point typewriter
amtt9	9-point typewriter
amsltt10	*10-point slanted typewriter*
amitt10	*10-point italic typewriter*
amcsc10	10-POINT SMALL CAPITALS
amu10	10-point unslanted italics
amdunh10	Tall thin letters for no known purpose
amnaro	Tall thin sans-serif letters

Bibliography

AMERICAN MATHEMATICAL SOCIETY "Differences between TEX82 and TEX80." In *TEX and Metafont: Errata and Changes*. Published for the TEX Users Group by the American Mathematical Society, Providence, RI. September 1983

FURUTA, R., SCOFIELD, J. and SHAW, A. "Document Formatting Systems: Survey, Concepts, and Issues." In *Computing Surveys*, **14**, 3 (Sept. 1982), 417-472.

KERNIGHAN, B. W. and CHERRY, L. L. "Typesetting Mathematics — User's Guide (Second Edition)." In *Unix Programmer's Manual*, Volume 2A. Western Electric Co., Greensboro NC 1976

KNUTH, D. E. "TEX, a system for technical text." In *TEX and Metafont: New Directions in Typesetting*, part 2. Digital Press and the American Mathematical Society, Bedford, MA, and Providence, RI. 1979.

KNUTH, D. E. *The TEXbook*. Addison-Wesley Publishing, Reading, MA 1983

LESK, M. E. and KERNIGHAN, B. W. "Tbl — A Program to Format Tables." In *UNIX Programmer's Manual*, Volume 2A. Western Electric Co., Greensboro NC 1976

OSSANA, J. F. "NROFF/TROFF User's Manual." In *UNIX Programmer's Manual*, Volume 2A. Western Electric Co., Greensboro NC 1974

REID, B. K. *Scribe Introductory User's Manual*. Unilogic Inc., Pittsburgh PA, 1980