TEX AND METAFONT: ERRATA AND CHANGES

As of 09 September 1983

This document contains all known corrections and changes to the AMS/Digital Press edition of the TEX/**METAFONT** manual (December 1979) and to the TEX and **METAFONT** programs themselves, as compiled by the TEX group at Stanford. The implementation of TEX described in that manual (TEX80) is frozen, and preliminary versions of TEX82 have been distributed.

This is the final issue of this errata list which will contain the TEX80 errata and extensions.

Differences between TEX80 and TEX82 are listed here in full, as of September 9, 1983; debugging is still underway, and subsequent changes are being logged at Stanford as indicated on page 12. An index to the TEX82/TEX80 differences, referenced by item number, begins on page 38.

CONTENTS

Distributed with TUGboat Volume 4, No. 2

Published for the TEX Users Group by the American Mathematical Society

P.O. Box 6248, Providence, R.I. 02940

TEX80 Errata

As of 09 September 1983

Changes made subsequent to the Digital Press edition of the manual (December 1979)

Page 8, the display at the bottom of the page was lost; it should say

␣

Page 19, line –4, change "`\vskip 1in`" to "`\topspace 1in`". And change the wording of page 20 so that it says that `\topspace` gives you space at the top of a page, `\vskip` gives space elsewhere (but `\vskip` is deleted when it appears at the top of any page). The control sequence `\topspace` is now defined in `basic.tex`.

Page 26, line –8, should be

```
\hsize 1 in \jpar 1000000 \ragged 1000
```

Page 27, lines 6 and 7, should be

```
...trline #1->\hbox to size{\hss #1
                                \hss }
```

Page 27, lines 14 and 15 are replaced by a single line

```
\hbox to size{\hss#1\hss}
```

Also delete "and followed" from line 19, and change "`\hskip` etc." on line 21 to "`\hss }`".

Page 39, line –9, change "`1.667pt`" to "`.33ex`"; on line –7, change "`2.5pt`" to "`.5ex`"; and replace lines –6 through –4 by:

"(This is for Computer Modern fonts. Other font designs may require different positioning; you will probably be able to find somebody who has worked out control sequences that you can use for any necessary accent on any particular alphabet.)"

Page 40, new units of distance:

`ex`	One "xheight" of space in the current font
`bp`	"big point" (one inch equals exactly 72 "big points")
`mi`	mica (one millimeter equals 100 micas)
`cc`	cicero (one cicero = 12dd); this is big in Europe

Page 41, add a new sentence after "devices." on line 13:

If you want to avoid this inflation factor, you can say "`true`" just before the unit; e.g., "`4truein`". (No space after "`true`".)

Page 50, line 7 should say "100000 points, which is about 115 feet;" and change "`1000cm`" to "`100000pt`" on lines 9 and 10.

Page 61, line –14, change "`\omega`" to "`\pi`", and the example to match;
line –13, change "`\varomega`" to "`\varpi`".

Page 92, line 12, change "formula is placed flush left" to:

formula is centered between the left margin and the equation number, unless glue appears at the beginning of the formula; if such glue has been explicitly specified, it is placed flush left.

Page 95, the two-line displays on this page should have had 2pt extra space between the lines.

Page 99, line 1, "within" should be "with";
line 16, delete the sentence "The dimensions you specify should not be negative."

Page 102, line 4, change to

```
\def\TeX{\hbox{\:aT\hskip-.1667em\lower.424ex\hbox{E}\hskip-.125em X}}
```

Page 116, line –3, insert "or \ " before the comma.

Pages 120 and 129, add a new entry to the list of internal parameters:

10	dumplength	500	Appendix X

Page 139, change lines –15 and –16 to read:
`! Argument of` ⟨control sequence⟩ `has an extra }.`
Some argument to the specified macro has more }'s than {'s. Proceed, and ...

Page 141, line 15, two new error messages:
`! Incomplete if.`
This means that `\if` is being applied to something other than two characters (e.g. `\if \hsize` ...). Proceed, the `\if` will be ignored.
`! Input page ended on different nesting level (`n`).`
This indicates that some page of a file has n more {'s than }'s (more precisely, that the level of nesting at the end of the page was n more than it was at the beginning of the entire file). You probably should insert n forgotten '}' symbols (if n is positive), or $|n|$ forgotten '{' symbols (if negative), before continuing, so that the balance is restored.

Page 142, lines 12–14, replace "But be careful: ... just now." by:
Remember that blank space after the name.

Page 151, use simpler definitions for `\rjustline` and `\ctrline`:

```
\def\rjustline#1{\hbox to size{\hss #1}}
\def\ctrline#1{\hbox to size{\hss #1\hss}}
```

Page 152, a new definition of `\twoline` replaces the old:

```
\def\twoline#1#2#3{\vbox{\hbox to size{$\quad\dispstyle{#1}$\hfill}
    \vskip#2\hbox to size{\hfill$\dispstyle{#3}\quad$}}}
```

Also we simplify the definitions of `\chop` and `\spose`:

```
\def\chop to#1pt#2{\save0\hbox{$\dispstyle{#2}$}\hbox
                  {\lower#1pt\null\vbox to 1ht0{\box0\vss}}}
\def\spose#1{\hbox to 0pt{#1\hss}}
```

And add a new definition:

```
\def\topspace{{\hrule height0pt}\vskip}
```

Page 163, line 16, put "}" after "⟨last line⟩`\cr`"

Page 170, ... The same codes are used for slanted roman fonts like cms10, and for text italic fonts like cmti10.

Page 171, line 3, "except for `\b`, `\l`, `\o`, `\t`, `\H`, and";
line 7, change the "except for" list to include underscore (_), and exclude the ≤ and ≥ signs.

Page 171, the typewriter type code now has an underline in position ´032, suitable for making ≤ and ≥ signs from < and >. Position ´033 is the German ß as on page 170.

Page 172, delete the sentence "The same codes apply ... manual)."

Page 172, codes ´134, ´136, ´137 become undefined.

Pages 173, 176, 178, a new binary relation ≍ (`\heqv`, Hardy equivalence symbol) now lives in position ´067 of the symbols font.

Page 177, change "`\varomega`" to "`\varpi`".

Page 180, the hyphenation algorithm has been improved.
line –11, change "350" to "310"; later on in that paragraph change "`in-form-ant`" and "`in-for-ma-tion`" to "`ex-press-ible`" and "`ex-pres-sion`", respectively. (Since some dictionaries accept "`in-for-mant`", it is no longer an exception!)

Page 180, line –2 ff., the suffix removal rules should now read:
A permissible hyphen is inserted if the word ends with `-able` or `-ably` (unless preceded by `c`, `f`, `g`, `p`, `r`, `et`, `it`, `ot`, `tt`, or `ut`), `-ary` (preceded by `ion`), `-cal`, ..., `-nary` (unless preceded by `io`), `-ogy`, ...

Page 181, line 7, change to read:
is preceded by three or fewer letters, break `-ing` only if one of the preceding letters is a vowel. Otherwise if `ing`

Page 181, line 13, insert "-ably" between "-able" and "-ary";
line 22, insert "f" between "c" and "h";
line 26, change "l, or m" to "l, m, p, s, or v".

Pages 182–186, The exception dictionary has been extensively changed; new exceptions since October 1, 1980, will not be listed here.

Pages 184–185, delete these words from the exception dictionary (since it turns out they aren't hyphenated badly after all): harangue, meringue, tongue, senseless, morgue, torque, unsearchable.

Page 186, line 8, "been also" should say just "also".

Page 186, new special exceptions: cat-e-go-ry de-vel-op prob-lem-atic pro-gram-ming ref-er-enc*e

Page 188r, cmti10, change "172" to "170";
delete the entry for cmu10;
add pp. 116 and 201 to the Control space entry.

Page 189r, new entry: Dump length, 201.

Page 190l, add pp. 116 and 201 to the Escape space entry;
new entry: \ETC, 201;
delete "Footnotes, 164."

Page 191l, new entry: \heqv ($\asymp$), 178.

Page 192l, the \lineskip entry: change 115 to 116.

Page 195r, add p. 102 to the "TEX logo" entry.

Page 196l, new entries: \topspace, 19–20, 152.
true, 41.

Page 196r, change the \varomega entry to \varpi, and move it to follow \varphi;
the \vfilneg entry, put ")" after "\vfil".

Page 197l, add pp. 116 and 201 to the "\␣" (escape space) entry.

Page 198, line 14, change "[1,3]" to "[TEX,SYS]".

Page 199, line –14, insert after "mode)": "or in restricted vertical mode";
line –13, delete "horizontal".

Page 201, last line, Chapter 2 should be Chapter 3.

Page 201, new features:

13. You can now use up to 64 different fonts (the 64 codes on page 14 give 64 distinct possibilities).

14. New one-character control sequence feature: If you use \chcode to change the code of a character to type 13 (see page 32 of the manual; previously code 13 was illegal), TEX will treat it as if it were immediately preceded by your escape character. For example, suppose you want to type several left-justified lines in a row. Here's an "easy" way to do this by redefining code ´15 (which is a carriage-return):

```
\chcode´15=13 \def
{\par}
the first line
the second line
\chcode´15=5
```

The effect will be the same as

```
the first line\par
the second line\par
```

after which carriage-returns are back to normal again. Similarly, \chcode´40=13 will change spaces into escape-spaces.

Actually there is one important difference between single-character control sequences and ordinary ones: Blank spaces are not ignored after single-character control sequences. (TEX is in "state *M*" after reading them, cf. pp. 30–31.)

15. \leqno is analogous to \eqno except that the equation number is placed at the left.

16. If you see "\ETC" at the end of a token list being displayed by TEX (e.g., an ⟨argument⟩ in an error message), that token list is more than 500 characters long. This number 500 can be changed using \chpar10=⟨desired value⟩.

****

TEX80: the last extensions to TEX before 1990

Here are some new primitives that seem to enhance the macro-writer's job, as well as some final cleanups being made to the language before it is frozen and widely described in PASCAL implementation.

1. \font ⟨fontcode⟩=⟨filename⟩ **AN INCOMPATIBLE CHANGE!**
(Allowed in any mode.) This should precede your first use of that font code. For example,

```
\font a=cmr10
```

now appears in file basic.tex. If this font code has already been defined, TEX will give an error message unless the file name is the same. (In the SAIL versions of TEX, no error message will be given if this is a pre-loaded font code, because the SAIL runtime routines forget all strings at the beginning of the program and TEX doesn't know the names of the preloaded fonts. Thus you must still be careful not to redefine the font codes used in Appendix B.)

You should still use \: to select the current font, since \font doesn't do that.

If you refer to a font code (with \: or \mathrm, etc.) that has not yet been defined, the error message "! Undefined font code." occurs. To recover, TEX will expect to see a font file name (possibly preceded by = or \underscore); so you can just hit carriage-return after this error message if you want the old (pre-\font) conventions of TEX.

2. \copy⟨digit⟩
(Allowed wherever TEX expects a ⟨box⟩.) This is just like \box⟨digit⟩ except that the value of \box⟨digit⟩ is not reset to null. For example,

```
\save1\copy0
```

makes \box1 a copy of \box0.

3. \unbox⟨digit⟩
(Allowed in vertical or horizontal mode but not math or display math mode.) The contents of \box⟨digit⟩ are "unglued" and appended to the current vertical or horizontal list. In horizontal mode, the space factor is set to 1; in vertical mode, no baseline correction will be made to the following box. The contents of \box⟨digit⟩ are reset to null. For example,

```
\save1\hbox{\unbox1\box0}
```

appends \box0 to the previous contents of \box1 and destroys \box0.
Note: You can't \unbox an \hbox in vertical mode or a \vbox in horizontal mode.

The potential uses of \unbox to produce multi-column format with balanced columns are somewhat frightening—they might open Pandora's box.

4. \open⟨digit⟩=⟨filename⟩
(Allowed in any mode.) This specifies a file to which your TEX program will be able to send characters as output. Up to ten such files may be open at once. If the ⟨digit⟩ you specify was previously used to open a file, that file will be closed immediately, so be careful. The purpose of \open is to enable nice macro packages to be written for indexes, tables of contents, cross-references, etc.

5. \send ⟨digit⟩{⟨mark text⟩}
(Allowed in any mode.) As with \mark, all macros in the given text will be expanded when TEX first sees this \send instruction, but \counts are not expanded, nor are the names of control sequences following \def or \xdef or \gdef. The text will eventually be written on the open file corresponding to ⟨digit⟩ (or on your terminal and on errors.tmp if no such file is open), at the time of output; at that time, \counts and \topmarks, etc. are expanded. For example, suppose you say

```
... Let us define a tree{\send 5{Tree, \count0.}} as ...
```

in some paragraph. The text "Tree, \count0." is invisibly attached to the word "Tree"; and if \count0 equals 25 when the formatted page containing this word is output, the string "Tree, 25."

will be sent (followed by ⟨carriage-return⟩ and ⟨line-feed⟩). If TEX control sequences appear in the text to be output, the result is the same as you get when lists of tokens are displayed in TEX error messages. Here's another example. Suppose you have said

```
\open 5=refs.tex
```

and at some later point you say when \count3 = 21

```
\xdef\cv{\count3}
\send 5 {\def\tarj{\cv}}
\advcount3
```

The result is to send "\def \tarj {21}" to file refs.tex, and to advance \count3 to 22. Your input file could say "\input refs" near the beginning; then the next time you TEX your file, a reference like "[\tarj]" will become "[21]". The actual sending takes place when the page containing the \send gets output; in this case \xdef was used to send the value of \count3 before it advanced.

6. \ifdimen ⟨dimen⟩ ⟨relation⟩ ⟨dimen⟩ {⟨true text⟩} \else {⟨false text⟩}
(Allowed in any mode.) Here ⟨relation⟩ is any of the three symbols <, >, =. For example,

```
\ifdimen .5wd3 > 2em{a}\else{b}
```

yields "a" if half the width of \box3 exceeds two ems in the current font, otherwise it yields "b".

7. \parval⟨number⟩ and \codeval⟨number⟩ can be used whenever TEX is expecting a ⟨number⟩; they specify the current value of a TEX parameter or code (the values that are changed by \chpar and \chcode).

8. \linebreak in horizontal mode forces a line break but not a page break (so it's like half of \eject). \pagebreak in horizontal mode forces a page break after the line that contains it, but not a line break (so it's like the other half of \eject).

9. \let ⟨control sequence⟩=⟨control sequence⟩
(in any mode) makes an alternate name valid for the present meaning of the control sequence on the right. For example,

```
\let\usefont=\:
```

makes it possible to redefine the control sequence \: (for example if you want to use : as a one-character control sequence). And

```
\let\finishpar=\par
\def\par{\finishpar\hangindent 40pt}
```

sets up hanging indentation after all paragraphs (including those terminated by double-blank lines).

10. \mskip ⟨dimen⟩ [plus ⟨dimen⟩] [minus ⟨dimen⟩]
in math mode gives glue that varies with the current style. The dimensions must be given in terms of a new unit "mu" that is allowed only with \mskip. One mu is 1/18 of a \quad in the current style. Thus, for example,

```
\mskip 5mu plus 5mu
```

gives the spacing of a "thick space" in display style (cf. p. 81).

11. New parameters for things that used to be frozen into the code:

\chpar11 = "radsign", the 18-bit code that will be used for \sqrt (same format as used in \left\char, see p. 79). Default is ´560760.

\chpar12 = "fudge", 1000 times the magnification factor used in computing "true" dimensions. Default is 1301 (XGP), 1100 (Dover), 1000 (at Stanford when we get rid of the XGP). The output routine might also look at this parameter to scale the output, although this isn't implemented yet and the desired scale will probably be indicated by a spooling command in most cases.

\chpar13 = "adjpen", penalty for adjacent lines being more than one step apart in the sequence (shrunk at least 50%, near normal, stretched at least 50%, stretched at least 100%). As in the case of double-hyphenation (\chpar3), you give the square of the desired penalty. Default is 3000.

\chpar14 = "loose", causes a paragraph to be set this many lines longer than the optimal number (if you want to avoid widow lines in straight text), if possible. For example, \chpar14=1 makes

subsequent paragraphs a line longer; `\chpar14=-1` makes them a line shorter (if there's a way). Default is, of course, 0.

`\chpar15` = "jjpar". The new TEX justification routine tries first to break lines in a paragraph with no hyphenations, using `jjpar` instead of `jpar` to ferret out unfeasible breaks. If this fails, or if "loose" is nonzero, another attempt is made, trying all possible hyphenations, and using `jpar`. Default is 2. (A high value makes TEX accept badly stretched lines before trying to hyphenate.)

`\chpar16` = "uchyph". If nonzero, TEX will attempt to hyphenate words containing upper case letters, using the algorithm of Appendix H. Default is 0.

`\chpar17` = "exhyph". The penalty for breaking after an explicit hyphen or dash (see rule e on p. 54). Default is 50.

`\chpar18,19,20`, reserved for communication with special extensions in private versions of TEX.

12. New glue parameters in addition to `\baselineskip`, etc.:

`\specskip` ⟨digit⟩, for any type of glue that you want to use often. This glue is accessible in any mode by writing `\skip` ⟨digit⟩; it saves internal memory space in TEX.

`\spaceskip`, if nonzero will be used for spaces instead of the values from the current font. If you say `\spaceskip .3em plus .4em`, the meaning of "em" at the time `\spaceskip` appears is used (it won't change when the em changes).

`\xspaceskip`, like `\spaceskip`, but this is the glue used for spaces following periods, question marks, exclamation points (i.e. space factor 3).

`\parfillskip`, is the glue automatically inserted at end of each paragraph. This is the only `\...skip` parameter that is initially nonzero; it is initialized to `\hfil`. (At paragraph end, TEX removes the last item of the current horizontal list if it is glue, under the assumption that it was the space that is scanned before a blank line or before `$$` causes the line-breaking routine to start. Then `\penalty1000` and the current `\parfillskip` glue are appended.) By giving this parameter nonzero space and controlling its stretchability and shrinkability, you can keep the final lines of your paragraphs from being too empty or too full.

13. `\parshape` $n\ i_1\ l_1\ i_2\ l_2\ \ldots i_n\ l_n$
where n is an integer and each i_j or l_j is a ⟨dimen⟩, takes precedence over hanging indentation (of which it is a major generalization). The first n lines of each subsequent paragraph will be of the specified lengths and will have the specified indentations; lines after the n^{th} will use the specifications for line n. To shut this off, type `\parshape 0`. (This feature will not work sensibly in connection with displayed equations, since TEX considers a displayed equation as if it breaks the paragraph into separate paragraphs.)

Note: A technical paper discussing all these bells and whistles of line breaking is being prepared by Knuth and Plass for publication in *SOFTWARE Practice & Experience*.

14. `\spacefactor` n, where n is an integer, can now be used in horizontal mode to set the value of the space factor to $n/100$. Previously a user had no way to override TEX's automatic conventions for this number.

Extensions to TEX since printing of errata, October 1, 1980

* Magnified printing and \chpar12:
It is sometimes valuable to be able to control the magnification factor at which documents are printed. For example, when preparing document masters that will be scaled down by some factor at a later step in the printing process, it is helpful to be able to specify that they be printed blown up by the reciprocal factor. There are several new features in TEX to allow for greater ease in the production of such magnified intermediate output.

TEX should be thought of as producing as output a "design document": a specification of what the final result of the printing process should look like. In the best of worlds, this "design document" would be constructed as a print file in a general and device independent format. Printing a magnified copy of this document for later reduction should be viewed as the task of the printer and its controlling software, and not something that TEX should worry about. But real world constraints force us to deviate from this model somewhat.

First, consider the plight of a TEX user who plans to print a document magnified by a factor of two on a printer that only handles 8.5″ by 11″ paper. In order to determine an appropriate \hsize and \vsize, this user will have to divide the paper dimensions by the planned magnification factor. Since computers are so good at dividing, TEX offers this user the option of setting the "magnification" parameter to 2000, warning TEX of the anticipated factor of 2 blow up, and then specifying \hsize and \vsize in units of "truein" instead of "in". When inputting a "true" distance, TEX divides by the scale factor that "magnification" implies, so as to cancel the effect of the anticipated scaling. Normal units refer to distances in the "design document", while "true" units refer to distances in the magnified printer output.

Secondly, some existing print file format and printer combinations have no current provision for magnified printing. A Press file, for example, uses absolute distances internally in all positioning commands, and Press printers treat these distances as concrete instructions without any provision for scaling. There is a program that takes a Press file and a scale factor as input and produces as output a new Press file in which all distances have been appropriately scaled. But it is inconvenient to be forced to use this scaling program on a regular basis. Instead, the Press output module of TEX chooses to scale up all distances by the "magnification" factor when writing the output Press file. Thus, the Press files that TEX writes are not representations of TEX's abstract "design document", but rather representations of the result of magnifying it by the factor (\parval12)/1000. On the other hand, the DVI files written by other versions of TEX containing normal units of distances, and the software that translates DVI files to instructions that drive various output devices will do the magnification by themselves, perhaps even using a magnification that was not specified in the TEX source program; if the user has not specified "true" dimensions, his or her DVI output file will represent the design document regardless of magnification.

Caveat: Due to the manner in which the current implementation of TEX writes Press files, it is not permissible to change the value of parameter 12 in the middle of a TEX run. If you want to produce magnified output, you should reset parameter 12 once very early in your document by using the \chpar12 control sequence, and from then on leave it alone. Another Caveat below discusses the situation in more detail.

* Magnified fonts, an extension to \font:
The magnification mechanism has been extended to include font specifications as well: in order to print a document that is photographically magnified, it is essential to use magnified fonts. A font is specified by the "\font" control sequence, which now has the syntax

\font ⟨fontcode⟩=⟨filename⟩ at ⟨dimen⟩.

The "at" clause is optional. If present, the dimension specified is taken as the desired size of the font, with the assumption that the font should be photographically expanded or shrunk as necessary to scale it to that size times the magnification factor specified by parameter 12. For example, the two fonts requested by the control sequences

```
\font a=CMR10 at 5pt    and    \font b=CMR5 at 5pt
```

will look somewhat different. Font a will be CMR10 photographically reduced by a factor of two, while font b will be CMR5 at its normal size (so it should be easier to read, assuming that it has been designed well).

The dimension in a font specification can use any units, either standard or "true". The interpretation of "true" here is identical to its interpretation in the specification of any other distance: asking for a font "at 5pt" requests that the font be 5 points in size in TEX's "design document", while asking for a font "at 5truept" requests that the font be 5 points in size after the scaling implied by the "magnification" factor.

If the "at ⟨dimen⟩" clause is omitted, TEX defaults the requested size to the design size of the font, interpreted as a design (non-"true") distance. Thus, the control sequence "\font a=CMR10" is equivalent to the sequence "\font a=CMR10 at 10pt", assuming that the designer of cmr10 has indeed told TEX that cmr10 is a 10-point font.

Caveat: This extension allows the TEX user to request any magnification of any font. In general, only certain standard magnifications of fonts will be available at most printers. The user of TEX at any particular site must be careful to request only those fonts that the printer can handle.

Caveat: As mentioned above, you shouldn't change the value of parameter 12 in the middle of a run. TEX uses the value of parameter 12 in the following three ways:

(i) Whenever the scanner sees a "true" distance, it divides by the current magnification.

(ii) At the end of every page, TEX's output module may scale all distances by the current magnification while converting this page to format for an output device.

(iii) At the very end of the TEX run, the output module uses the current magnification to scale the requested sizes of all fonts.

Given this state of affairs, it is best not to change parameter 12 once any "true" distance has been scanned and once any page has been output.

* A new feature to solve the 'first footnote on a page' problem at last!

```
\topsep{⟨restricted vertical mode stuff⟩}
and
\botsep{⟨restricted vertical mode stuff⟩}
```

will insert their 'stuff' between all \topinserts or \botinserts on a page and the text on that page, unless no insert is present. For example, suppose a page has two \topinserts and one \botinsert, then you get

```
\topinsert #1
\topskip glue
\topinsert #2
\topskip glue
\topsep
the text of the page
\botsep
\botskip glue
\botinsert #1.
```

A \topsep or \botsep specification is global and must appear in unrestricted vertical mode.

****

Extensions to TEX since printing of errata, February 4, 1981

* I might include the following quote from George Bernard Shaw in the next edition of the user manual, near page 99:

"The only thing that never looks right is a rule.
There is not in existence a page with a rule on it that cannot be
instantly and obviously improved by taking that rule out."
in *The Dolphin* 4 (1940), p. 81.

* Various TEX constructions use words that are not control sequences, namely

```
after  at  by  depth  expand  for  height
minus  par  plus  size  to  width
```

and dimensions like `pt`, `em`, etc. Such words will now be recognized if they use uppercase letters, e.g.,

```
\hbox Par SIZE{\hskip 1EM PLUS 3dD}
```

* Replying with "form feed" to an error message is like "line feed" except more drastic: Nothing will stop TEX from then on until completion of the run. (There will be no more "*" or "^" prompts, and tracing won't pause either.)

* `\dpenalty` is now allowed in display math mode. It specifies the penalty for a page break before the glue that follows this display.

* `\mark` is now allowed in horizontal mode.

* An optional space can precede { in contexts where { is required, e.g. "`\if AA {`".

* Glue and penalties at the top of a page will be deleted after marks and sends.

****

Extensions to TEX since printing of errata, June 30, 1981

* `\xleaders` and `\cleaders` are new kinds of nonaligned leaders. Suppose the space to be filled by leaders has length s and the box that fills it has size b. Then if b goes at most q times into s, you get either q or $q-1$ appearances of the box when you use TEX's `\leaders`. With `\xleaders` and `\cleaders` you always get q copies of the box, but they don't necessarily line up with other leaders on the page. Let $r = s - qb$ be the remaining space. Then `\xleaders` (expanding leaders) puts $r/(q+1)$ of the extra space between each box and at the ends; `\cleaders` (centered leaders) puts $r/2$ of the extra space at each end and butts the boxes tightly together.

* **Incompatible change** to typewriter fonts (page 171 of the manual): The characters in positions ´175 and ´176 have switched places.

* You can now use most single characters where TEX expects a ⟨number⟩; the result is the ascii code of that character. (This extension was permitted for letters in a previous notice in this errata file. Now it is permitted also for characters of type "otherchar" as well (see Chapter 7), as long as the otherchar is not a digit or a ´ mark or a + or - sign or a . that might precede a digit. For example, "`\chcode }=2`" is now a good thing to say in the file `BASIC.TEX`. Other characters can be used too except in cases where the scanner pre-empts them (namely, escape characters, end-of-line characters, ignored characters). A new error message is given if no legal number is present:

```
! Improper number, 0 inserted.
```

(Previously, 0 was inserted without warning.)

* To test whether two control sequences are equal, say

`\ifx` ⟨cs1⟩⟨cs2⟩{⟨true text⟩}`\else`{⟨false text⟩}

For example, "`\ifx\foo\bar{yes}\else{no}`" will give "yes" if `\foo` has been defined to be the same as `\bar`, using `\def` or `\let`. This means the definition really is the same, not that it ultimately expands the same. Examples: Consider the definitions

```
\def\foo{abc} \gdef\bar{abc} \def\bah{\bar} \xdef\blah{\bar}
  \def\empty#1{} \def\empt{} \let\emp=\empt \let\goo=\ifx
```

and suppose that \und and \new have never been defined. Then

\ifx\foo\bar is true
\ifx\bar\bah is false
\ifx\foo\blah is true
\ifx\und\new is true
\ifx\und\empty is false
\ifx\emp\empt is true
\ifx\emp\empty is false (since the parameters are different)
\ifx\goo is true
\ifx\emp\def is false
\goo\emp{} gives "! Two control sequences must follow \ifx."

This feature can be used to tell if a control sequence has been defined. (For example, if definitions are read from a cross-reference file, that file is empty on the first iteration, so a cross-reference macro needs to do something different the first time.) It has also proved useful for making indexes (recognizing when two strings of characters are equal, so they specify the same index entry).

* New changes to the Computer Modern Fonts have caused the TEX logo to change a bit. Here is the recommended control sequence (for page 102):

```
\def\TeX{\hbox{\:aT\hskip-.125em\lower.5ex\hbox{E}\hskip-.075em X}}
```

* Position ´167 in cmsy fonts is now occupied by a diamond operator that goes with the circle and bullet. It's called \diam.

* When responding "?" to an error, you now get a different help-message:

```
Type c or C to continue, s or S to scroll all error messages,
etc.
```

So you can get these effects without control codes on various terminals. The old ⟨cr⟩ (continue) and ⟨lf⟩ (scroll) still work. There's also ⟨ff⟩ or f or F, which sets up nonstop mode; the latter scrolls errors and doesn't pause for any reason. On TOPS20, you can get into nonstop mode by typing a slash at the end of your command line (e.g. "@tex foo/"). Nonstop mode is intended for overnight batch processing.

* Wizards who try to do weird things with horizontal and vertical lists need one more feature, which DEK promises is the last feature to be added before TEX becomes frozen and solid. A new type of box specification is allowed: If you are in (possibly restricted) horizontal or vertical mode, and if the last thing in the current horizontal or vertical list is a box, "\thebox" denotes this box, and removes it from the list. Otherwise \thebox denotes nothing. For example, if \box1 contains two boxes separated by glue, you can pick it apart by saying

```
\unbox1 \save2\thebox \unskip \save1\thebox
```

* Slight change to \halign and \valign: Each column after an \halign now has the height and depth of the entire row, and each row after a \valign now has the width of the entire column. This new interpretation (which is the way alignment always should have worked!) doesn't change anything except horizontal and vertical rules that extend to the edge of their containing boxes. As a result, it is *much* easier to make vertical rules in \haligned tables. (But one should still use \baselineskip 0pt and \lineskip 0pt and some form of strut to avoid gaps between the lines.)

* A new type of dimension is now allowable in the "plus" and "minus" parts of glue specifications, namely "fil", "fill", and "filll". For example, \hfil and \hfill are respectively equal to \hskip 0pt plus 1fil and \hskip 0pt plus 1fill; \hss is the same as \hskip 0pt plus 1fil minus 1fil. In the present TEX, the respective meanings of fil, fill, and filll are 10^5, 10^{10}, and 10^{15} points. However, next year's TEX will treat these as infinity, infinity^2, and infinity^3, and the coefficients of powers of infinity will be limited in absolute value. Therefore, for compatibility, present users of TEX should not include large amounts of stretch and shrink in glue specifications unless they use the new "fil" feature. TEX will give you a warning message suggestion that you use "fil" if the magnitude of a stretch or shrink component exceeds 32767pt. For example, if you have a TEX program that says "\topspace 10pt plus 100000pt", you should change it to "\topspace 10pt plus 1fil" to avoid the warning message. See the next section for further details about next year's TEX.

Differences between TEX82 and SAIL TEX (TEX80), as of September 9, 1983

This file describes differences between TEX82, which is the portable standard definition of TEX-in-Pascal, and the TEX systems written in SAIL. (The SAIL version will *not* be brought up to date to make it compatible with TEX82; its use should gradually die away as more people take advantage of the new features available in the PASCAL version.)

Note: This list is equivalent to the file `TEX82.DIF[tex,dek]` on the SAIL system at Stanford, as of the date shown. While it contains much of the same information as the lists published in earlier issues of TUGboat, the order of the items may be different, and redundancies and obsolete material have been removed. Old lists should be disposed of.

(1) TEX82 does all its calculations that affect line breaking and page breaking using fixed-point integer arithmetic of limited (i.e., 32-bit) precision, instead of with floating-point computations, since different machines differ so widely in the results you get with floating point.

Dimensions are integers in units of 2^{-16} points, limited in magnitude to 2^{14} points (which is 18.89 feet). This applies to all dimensions (e.g., the heights and widths and depths of boxes, the amounts by which you `\raise` or `\lower` a box, `\varunits`, etc.), except for the dimensions of stretching and shrinking.

Something different had to be done with respect to the dimensions of stretching and shrinking, since for example the old TEX defined `\hfill` to be a stretch of 10^{10} points, and that number has more than 32 bits to the left of its binary point. After considering various alternatives, the solution introduced by the designers of MESA-TEX in 1979 has been adopted for TEX82. Each stretch or shrink dimension is specified by a fixed point integer that is either in units of 2^{-16}`pt` or 2^{-16}`fil` or 2^{-16}`fill` or 2^{-16}`filll`. Here `pt` is, of course, one point; the other units are three orders of infinity, essentially infinity and infinity^2 and infinity^3. To add together such units of stretching or shrinking, one simply adds the individual components having the same order of infinity, and then uses the nonzero component having the highest order.

Thus, when one says "`\hskip` ⟨*a*⟩`pt plus` ⟨*b*⟩`fil minus` ⟨*c*⟩`filll`", the numbers ⟨*a*⟩, ⟨*b*⟩, ⟨*c*⟩ are rounded to the nearest multiple of 2^{-16}, and their magnitudes should be less than 2^{14}. The stretch component is ⟨*b*⟩ times infinity, and the shrink component is ⟨*c*⟩ times infinity cubed.

(2) The shrink component of all glue used in a paragraph should be finite. Something like "`\hskip 20pt minus 2fil`" actually makes no sense in a paragraph, since the paragraph would fit on a single line no matter what. Infinite shrinkage does make sense in a simple `\hbox`, of course.

(3) Identifiers for control sequences in TEX82 are letter strings of any length, with upper and lower case letters treated as distinct even when they aren't the first letter. For example, `\TeX` is not the same as `\TEX`, and `\GAMMA` is not the same as `\Gamma`. Any character that is regarded as a letter (this means the 52 letters, initially, plus others that are `\catcoded` to 11) can appear in such control sequences. Of course, the one-character non-letter control sequences still exist as well.

(4) Characters by themselves can no longer masquerade as numbers. Thus, you shouldn't say "`\catcode A`" any more; this has caused more mysterious errors than it was worth, and anyway there will be a macro for defining fonts symbolically in TEX82. Instead, one can use a left quote in front of a character to get its ascii code; e.g., `` `A `` is *´101*, and `` `↑↑A `` is 1. Furthermore you can use an escape in front of the character, e.g., `` `\A `` or `` `\% `` (these macros will not be expanded), so that we avoid the problem in TEX80 that you couldn't say "`\catcode %`" when % had already been catcoded.

(5) Hexadecimal constants are allowed, using a prefixed `"`. For example, `"DF` = *´337* = 223.

(6) `\uccode` ⟨char number⟩ is the character code to use when converting to upper case in the `\uppercase` function. For example, `\uccode`a =`A`. (Probably nobody will change the default values unless a foreign alphabet is being used.) Characters whose uccode is zero will not be changed by the `\uppercase` function.

(7) `\lccode` ⟨char number⟩ is the character code to use when converting to lower case in the `\lowercase` function or when trying to hyphenate a word that contains upper case letters. For example, `\lccode`A =`a`. Characters whose `lccode` is zero will not be changed by the `\lowercase` function. The `lccode` is also used for hyphenation; a character whose `lccode` is zero will be regarded as a nonletter (e.g.,

punctuation), and not part of a hyphenatable word, while a character whose `\lccode` is identical to the character itself will be considered a lower case letter. For example, `\lccode`a =`a`.

(8) If `\uchyph` is nonzero, a word whose first letter is upper case will be subject to hyphenation. (This means a word whose first letter has `\lccode` nonzero and `\lccode` not equal to the character itself.) Words whose first letter is lower case will always be subject to hyphenation even if they contain upper case letters further on.

(9) Both `\looseness` and `\parshape` are now reset to their default values after each paragraph, as `\hangindent` always was. They are also reset at the beginning of a `\vbox`.

(10) Font codes are now control sequences instead of letters, so there is no longer a 64-font restriction. For example, one says "`\font\ff=cmr10 at 12pt`" to load the font identified by `\ff`. A subsequent appearance of the control sequence `\ff` will select this font. (The `\:` control sequence is no longer present in TEX82.) Font control sequence names must have length 2 or more.

(11) New input capabilities greatly expand the class of potential applications:

`\openin` *n*`=filename`, where the = is optional and where the stream number *n* is between 0 and 15, designates a text file that can be read concurrently with other input. If a file has already been opened for the same number *n*, it is closed first. If the file doesn't exist, no file will be opened.

`\closein` *n* simply closes stream *n*.

`\ifeof` *n* ... `\else` ... `\fi` tests if input stream *n* is either not open or has been fully read.

`\read` *n* `to\cs` and `\global\read` *n* `to\cs`: These are the important new commands. They define control sequence `\cs` to be the contents of the next line from input stream *n*, reading that line in the normal way: the current catcodes are used, spaces are ignored at the beginning of the line, a blank line comes through as `\par`, etc. If input stream *n* has not been opened, or if an opened file has been fully input, or if *n* is negative or greater than 15, a line is read from the terminal instead; the user is prompted "`\cs=`" in this case. The latter feature allows convenient interactive routines, e.g.,

```
\message{Please type your name:}
\read-1 to\myname
\message{Hello, \myname!}
```

(12) The previous commands `\open` and `\send` are renamed `\openout` and `\write`, to show their similarity to `\openin` and `\read`.

`\closeout` ⟨number⟩ will close a file so that your TEX program can eventually input it. (Previously you could only do this by, e.g., `\openout 0=empty0.tmp`, cluttering up the disk with an empty file.)

`\write`, `\openout`, and `\closeout` will be ignored if they occur within a `\leaders` construction. (Reason: The number of times a leader box occurs might be 0 or 1 depending on floating-point rounding. This restriction keeps the language independent of floating point.)

(13) `\ifx` now allows comparison of any two tokens, not just control sequences. A non-control-sequence matches only another non-control-sequence that has the same catcode and represents the same character.

(14) New feature `\expandafter` *t*, where *t* is any token (usually a control sequence): If the token following *t* is a macro, it is expanded as if *t* were not there. Then *t* is put back in front of the result. For example,

```
\def\a{\x\y\z} \def\b#1\z{#1} \expandafter\b\a
```

yields `\b\x\y\z` which yields `\x\y`. (It used to be possible to do this only with trickery/hackery.)

(15) `\chpar` has been abolished. In its place, all of the integer parameters have names instead of numbers. (Thus at last they become consistent with the dimension parameters `\hsize`, etc., and with the glue parameters `\baselineskip`, etc. My only excuse for bad design in the first place was that the integer parameters were afterthoughts, stuck in after TEX was first up and running; it was the easiest way to

vary some of the originally fixed constants. I wanted to finish TEX in a year and get on to writing Volume 4! That is still my wish.)

Here are the names of the integer parameters:

\tolerance	(formerly \chpar1=)	badness tolerance after hyphenation
\pretolerance	(formerly \chpar15=)	badness tolerance before hyphenation
\hyphenpenalty	(formerly \chpar2=)	hyphenation penalty
\finalhyphendemerits	(formerly half of \chpar3=)	penultimate line hyphenation demerits
\doublehyphendemerits	(formerly half of \chpar3=)	double-hyphen demerits
\widowpenalty	(formerly \chpar4=)	widow line penalty
\brokenpenalty	(formerly \chpar5=)	broken line at page end penalty
\binoppenalty	(formerly \chpar6=)	math binary op break penalty
\relpenalty	(formerly \chpar7=)	math relation break penalty
\predisplaypenalty	(formerly \chpar9=)	penalty for breaking before a display
\mag	(formerly \chpar12=)	1000 × magnification ratio
\adjdemerits	(formerly \chpar13=)	adjacent incompatibility demerits
\looseness	(formerly \chpar14=)	change in paragraph length
\uchyph	(formerly \chpar16=)	uppercase hyphenation
\exhyphenpenalty	(formerly \chpar17=)	explicit hyphenation penalty
\day	(new)	initialized to current day of month
\month	(new)	initialized to current month of year
\year	(new)	initialized to current year
\time	(new)	initialized to minutes since midnight
\interlinepenalty	(new)	see below (*#94*)
\postdisplaypenalty	(new)	see below (*#143*)
\displaywidowpenalty	(new)	see below (*#143*)

The former \chpar10 (dump window) is no longer needed, since TEX82 has better ways to display token lists. The former \chpar11 (\radsign) goes away in favor of the new \radical primitive. The former \chpar18, \chpar19, \chpar20, once "reserved for extensions", are gone too, since it is now best for a TEX extender to give names to whatever new parameters are needed. Similarly, \x is gone.

(16) The \tracing parameter disappears, and its various components each have their own names. They are as follows:

\showboxbreadth	(nodes per level when a box is being exhibited)
\showboxdepth	(maximum level shown when a box is being exhibited)
\pausing	(if nonzero, lines from a file are displayed as they appear)
\tracingonline	(if nonzero, diagnostic info goes to terminal as well as to file)
\tracingmacros	(if nonzero, shows macros as they are being expanded)
\tracingstats	(if nonzero shows memory usage when TEX has recorded it)
\tracingoutput	(if nonzero, shows boxes when they are shipped out)
\tracinglostchars	(if nonzero, shows chars dropped because they aren't in font)
\hfuzz	(a dimen parameter; hboxes are reported if more overfull than this)
\vfuzz	(a dimen parameter; vboxes are reported if more overfull than this)
\hbadness	(under/overfull hboxes exceeding this (integer) badness are reported)
\vbadness	(under/overfull vboxes exceeding this (integer) badness are reported)

(17) The new \tolerance and \pretolerance are devalued by a factor of 100 from the old \jpar and \jjpar. In other words, the default is now \tolerance 200 and \pretolerance 200, so that it is a true "badness tolerance", i.e., the badness should not exceed 200. Any value of 10000 or more is equivalent to an infinite value, in which glue can stretch arbitrarily far.

(18) Furthermore, \codeval and \parval are eliminated. In their place is a much more powerful operator called \the. For example, what used to be "\codeval5" is now "\the\catcode5"; what used to be "\parval2" is now "\the\hyphenpenalty". You can even say "\the\hsize" to get the current \hsize as a dimension, "\the\baselineskip" to get the current \baselineskip as a glue value; and you can say things like "\vbox to \the\baselineskip", which makes a vbox whose height is the

normal amount of baselineskip (exclusive of stretching and shrinking). When expanding macros, you can say "\the\font" to get the current font identifier (e.g., "\ff"), as well as "\the\ff" to get the corresponding font name, as well as "\the\output" and "\the\everypar". In the latter cases, macros inside the current output or everypar routines are not further expanded.

(19) Things like \count and \dimen and \skip should not appear except in the context of numbers, dimens, and glue; for example, you shouldn't say "\dimen 5" or even "\count 5" in the midst of a paragraph. But if you say "\the\count5", the paragraph will get the (signed decimal) value of the counter; if you say "\the\dimen10" you will get text like "3.14159pt"; and "\the\skip12" yields text like "−5.00000pt plus 3.40001fil". In \xdef and \write, TEX82 will expand occurrences of \the, using the current values, but \count and \dimen and \skip will not be expanded. This gives you a little more control over what gets expanded.

(20) \thebox is changed to \lastbox (avoids confusion with \the).

(21) \minusthe is like \the, but gives the negative value.

(22) An array of 256 dimension values is introduced, called \dimen0 to \dimen255. Furthermore '3.5dm8' is a dimension equal to 3.5 times \dimen8. These join \count0 to \count255 and \skip0 to \skip255, so we now have 256 of each basic quantity (instead of 10, as in TEX80). This applies to \count, \dimen, \skip, \write, and \box. PLAIN.TEX contains macros for allocating a new \count or \dimen, etc.

(23) mu-glue and ordinary glue are now unmixable, since the rules are so much cleaner and clearer this way: There are 256 \muskip registers, which take glue whose units are mu (instead of pt, etc.). It is now illegal to do things like \hskip\the\thinmuskip or \mskip\the\baselineskip.

(24) Operations on \count, \dimen, \skip, and \muskip are extended, so that we now have a complete set:

\setcount ⟨digit⟩ [=] ⟨number⟩
\advcount ⟨digit⟩ by ⟨number⟩
\multcount ⟨digit⟩ by ⟨number⟩
\divcount ⟨digit⟩ by ⟨number⟩

\setdimen ⟨digit⟩ [=] ⟨dimen⟩
\advdimen ⟨digit⟩ by ⟨dimen⟩
\multdimen ⟨digit⟩ by ⟨number⟩
\divdimen ⟨digit⟩ by ⟨number⟩

\setskip ⟨digit⟩ [=] ⟨glue⟩
\advskip ⟨digit⟩ by ⟨glue⟩
\multskip ⟨digit⟩ by ⟨number⟩
\divskip ⟨digit⟩ by ⟨number⟩

\setmuskip ⟨digit⟩ [=] ⟨mathglue⟩
\advmuskip ⟨digit⟩ by ⟨mathglue⟩
\multmuskip ⟨digit⟩ by ⟨number⟩
\divmuskip ⟨digit⟩ by ⟨number⟩

Note that \specskip has changed its name to \setskip. The division operations truncate towards zero. The = sign in \setcount, \setdimen, \setskip, \catcode, \font, \openout, \let, and in other similar things, is now optional. Furthermore an equals sign is optionally allowed now after \tolerance, \hsize, \baselineskip, \setbox (the new name for \save), etc.

(25) If you put \global in front of \def or \let or \catcode or \tolerance or \baselineskip or even \parshape or \hangindent or a font identifier, the definition will now be global. Otherwise the definition is local (except for \gdef and \xdef). This is a change in the case of dimension parameters: \varunit, \parindent, \lineskiplimit, \mathsurround, \maxdepth, \topbaseline, \hsize, \vsize; you should put \global in front of these to get the former behavior. You probably wanted the former behavior only when changing \hsize or \vsize in an \output routine.

(26) \edef is a local \xdef. Both \edef and \xdef can now take arguments like \def and \gdef.

(27) Two other modifiers can be placed in front of `\def`, `\gdef`, `\edef` and `\xdef`:

`\long` means that the arguments to the macro are allowed to contain `\par` tokens; formerly this was always allowed, but now it is permitted only for "long" macros. Otherwise TEX will now stop when it sees `\par` going into an argument, presuming that a right brace was forgotten. This detects one of the most frequent errors made by TEX users, before it propagates to overflow the memory.

`\outer` means that the macro being defined is not allowed to appear subsequently either in an argument or in the right-hand side of a definition or `write` text, or in the preamble of an alignment. In other words, the macro should appear only at "quiet" times. This is another way to catch missing braces before too much damage is done. It used to be applied at the end of every page, but most TEX users don't use a page-oriented editor like E; therefore TEX82 does not treat file pages as an integral part of its control structure.

(28) `\begingroup` and `\endgroup` provide an alternate way to enter and leave groups for locally defined values. A `\begingroup` will not match a }, nor will { match `\endgroup`; the former gives the message "`Missing \endgroup inserted`" when the } occurs, and the latter inserts a "missing" }. Note that you can introduce `\begingroup` in one macro and `\endgroup` in another.

(29) If you say `\message{text}`, the terminal will display " `text`" immediately. For example, the new version of `PLAIN.TEX` contains message statements so that when `INITEX` inputs the file your screen looks something like this:

```
(PLAIN.TEX preloading macros, fonts, codes, hyphenation)
```

instead of "`(plain.tex 1 2 3 4 5 6)`". (Note that you don't say `\input basic` any more, and `PLAIN.TEX` is already preloaded when you run TEX, as explained below.)

Here's an example of a macro that displays names of sections when you get to them in a paper you are TEXing:

```
\outer\def\section#1{\vfill\eject\message{#1}\centerline{\bf#1}}
```

(30) You can also say `\errmessage{text.}`, which causes a TEX error message like

```
! text.
```

(31) `\catcode` ⟨n⟩ replaces `\chcode` ⟨n⟩. `\mathcode` ⟨n⟩ replaces `\chcode` ⟨$n+128$⟩.

(32) A new catcode value 14 denotes a character that is better for comments than the old code 5. A character of code 14 denotes end of the current line (i.e., ignore the remainder of that line), without inserting a blank space, and without considering that line to be all blank. Thus, if `%` is assigned type 14, you can have lines that are completely comments by starting them with `%`, without having this line come out as `\par`; and you can also end a line with `%` without having a blank space inserted there.

(33) Another new catcode value, 15, denotes an invalid character. When such a character is input, TEX82 issues an error message.

(34) Here's something that was *not* put into TEX82: It wouldn't be hard to make TEX understand `\escape` to mean 0, `\opengroup` to mean 1, ..., `\active` to mean 13, `\comment` to mean 14, and `\invalid` to mean 15; then you could say, e.g., `\catcode´14=\active` in the example above. But it seems wrong to make `\catcode` too easy, since that will only encourage more people to fiddle with the `\catcode` table. Let's leave this a black art, to be resorted to only with reluctance in times of emergency.

(35) You *can* do certain things now in horizontal mode, e.g., `\vfill`; TEX82 will silently insert the `\par` you forgot.

(36) `\discretionary{#1}{#2}{#3}` makes discretionary characters other than hyphens. It means the text should either contain #3 without a break, or else it should contain #1, then a break, and then #2. For example, `\-` is equivalent to `\discretionary{-}{}{}`. The parameters #1, #2, and #3 may not contain anything but letters and otherchars (not spaces or penalties, etc.); they need not all be in the same font, and TEX will insert ligatures and kerns within them if necessary. For example, the correct way to specify the hyphenation of "difficult" is "`di\discretionary{f-}{fi}{ffi}\-cult`". In German, the correct way to specify hyphenation of "backen" is "`ba\discretionary{k-}{k}{ck}en`"; presumably if we were doing a lot of these we would define a `\ck` macro so that one could type "`ba\ck en`" or "`ba{\ck}en`". The third part of a `\discretionary` must be empty, in math mode.

(37) The internal character set used by TEX82 is the same regardless of the external character set. There is no longer a difference like "\catcode`176" for right brace that applies only at SAIL! Right braces and underlines and tildes and notequals and a few others have been a source of problems that have now gone away. Furthermore there is now a way to input an ascii control character to any version of TEX82 by typing, e.g., ↑↑A.

TEX82 assumes that all of the standard ascii characters, shown in positions *040* through *176* below, are available; these characters are always converted to their standard ascii codes. For example, a TEX user who types A is asking for character 65 of the current font, even though the A might have entered the computer in EBCDIC or some other code. Non-standard-ascii characters might also be readable on some implementations of TEX. In such cases they should have the significance stated below, for best results; and all characters that cannot be converted to a compatible TEX code should be converted to *177*.

ascii	TEX	description	catcode	mathcode (plain TEX)
000	↑↑@	null	ignore	bin401
001	↑↑A	downarrow	submark	rel443
002	↑↑B	alpha	other	ord213
003	↑↑C	beta	other	ord214
004	↑↑D	and	other	bin536
005	↑↑E	not	other	ord472
006	↑↑F	epsilon	other	ord217
007	↑↑G	pi	other	ord231
010	↑↑H	backspace, lambda	ignore	ord225
011	↑↑I	tab, gamma	space	ord215
012	↑↑J	linefeed, delta	ignore	ord216
013	↑↑K	uparrow	supmark	rel442
014	↑↑L	formfeed, ±	endline	bin406
015	↑↑M	carriage-return	endline	bin410
016	↑↑N	infinity	other	ord461
017	↑↑O	partial	other	ord245
020	↑↑P	subset	other	rel432
021	↑↑Q	superset	other	rel433
022	↑↑R	intersection	other	bin534
023	↑↑S	union	other	bin533
024	↑↑T	for-all	other	ord470
025	↑↑U	there-exists	other	ord471
026	↑↑V	circle-times	other	bin412
027	↑↑W	leftrightarrow	other	rel444
030	↑↑X	leftarrow	other	rel440
031	↑↑Y	rightarrow	other	rel441
032	↑↑Z	notequal	other	rel434
033	↑↑[	escape, diamond	action	bin567
034	↑↑\	less-or-equal	other	rel424
035	↑↑]	greater-or-equal	other	rel425
036	↑↑↑	equivalence	other	rel421
037	↑↑_	or	other	bin537
040		space	space	ord464
041	!	exclamation	other	close041
042	"	double-quote	other	active "
043	#	hashmark	param	ord561
044	$	dollar-sign	math	ord577
045	%	percent-sign	comment	ord045
046	&	ampersand	align	ord046
047	'	apostrophe	other	active '

050	(	left-parenthesis	other	open050
051	)	right-parenthesis	other	close051
052	*	asterisk	other	ord052
053	+	plus-sign	other	bin053
054	,	comma	other	punct054
055	-	hyphen, minus-sign	other	bin400
056	.	period	other	ord056
057	/	slash	other	ord057
060	0	zero	other	var060
. . .				
071	9	nine	other	var071
072	:	colon	other	rel072
073	;	semicolon	other	punct073
074	<	less-than-sign	other	rel074
075	=	equal-sign	other	rel075
076	>	greater-than-sign	other	rel076
077	?	question-mark	other	close077
100	@	at-sign	other	ord574
101	A	uppercase-A	letter	var301
. . .				
132	Z	uppercase-Z	letter	var332
133	[	left-bracket	other	open133
134	\	backslash	control	bin404
135	]	right-bracket	other	close135
136	↑	caret	supmark	ord017
137	_	underline	submark	ord465
140	`	reverse-apostrophe	other	open140
141	a	lowercase-a	letter	var341
. . .				
172	z	lowercase-z	letter	var372
173	{	left-brace	open	open546
174	|	vertical-line	other	ord552
175	}	right-brace	close	close547
176	~	tilde	other	rel430
177	↑↑?	invalid	invalid	ord573

As before, the mathcodes (which replace Appendix F8 of the old TEX manual) are relevant only when the catcode is letter or other. (See below for the new catcode values.) Two possibilities are given for codes *010*, *011*, *012*, *014*, *033*, *055*; at most one of these should be chosen, and if both are present on some system keyboards the other should probably be disallowed for TEX input (mapped into *177*). However, since a user can change any catcode and any mathcode, strict conformity with these interpretations isn't absolutely necessary. To convert a file into a format that all TEXes can read, one should change null into ↑↑@, downarrow into ↑↑A, and so on. If a character set contains uparrow but not caret (e.g., the SAIL system falls into this category), the uparrow should be considered an ascii caret; code *013* will be used only if both uparrow and caret are present, as they are at MIT. Incidentally, this internal coding scheme is based on a scheme used at MIT, since the MIT code is faithful to ascii while allowing additional visible characters that are extremely convenient.

An appearance of ↑↑A is equivalent to an appearance of ascii code *001*, if the current catcode of ↑ is supmark. In particular, if somebody in a foreign country with more than 26 letters in the local alphabet wants to make \catcode ↑↑A = letter, then control sequences like \a↑↑A↑↑Ab (a four letter word) are permissible.

TEX82 puts *015* (ascii carriage-return) at the end of each line, except for the lines that are inserted with "i" after error messages. If the final character of the line is currently catcoded to be an escape

character (e.g., if you end an error-insertion with \, or if you do \catcode`15=0), the result is equivalent to \csname\endcsname (control sequence of length zero).

Of course, users are expected to type \ne instead of ↑↑Z if their system's character set doesn't contain a not-equal sign; TEX82 recognizes ↑↑Z as ascii *032* primarily to make it possible for straightforward translation of TEX files from one system so that they will work on another.

Some files contain ascii *014* (form-feed) characters as page marks. Such characters are ordinarily treated like carriage-returns, since the initial catcode for *014* is endline. In order to get TEX82 to do the error checking at the end of a page, as the old TEX did, you can say

```
\catcode`14=13 \outer\def↑↑L{\par}
```

(38) Note that the backslash character is now predefined as an escape character when TEX82 begins. The old idea about letting the user's first nonblank character be the escape has been abandoned. Furthermore TEXPRE has been replaced by a version of TEX called INITEX that allows an entire macro package to be preloaded; this macro package can define its own catcodes and mathcodes. The normal version of TEX already has "PLAIN.TEX" preloaded; the normal version of AMS-TEX already has the AMS-TEX macros and fonts preloaded.

The new rule for starting TEX is this: When you're running INITEX or a version of TEX that has a preloaded format, you can request format file f.fmt by typing '&f' after the ** prompt. When you're running VIRTEX, format file plain.fmt will be loaded unless you type '&somenonplainformatname' after the ** prompt. Thus, for example, the following ways of starting TEX are equivalent at SAIL and similar sites:

```
tex paper
r tex;paper
r tex
** paper
r tex
**\input paper
```

(The asterisks here are TEX's prompt character. On TOPS-20 what used to be "@tex paper/", indicating batch mode, is now "@tex \batchmode\input paper".)

(39) There are four new primitives \batchmode, \nonstopmode, \scrollmode, \errorstopmode that represent increasing amounts of interaction. \batchmode and \nonstopmode will never stop for any reason; \batchmode omits printing anything on the terminal (but the .log file gets everything, as usual). These nonstop options are intended for overnight batch processing. \scrollmode doesn't stop for error messages, but it does stop if files can't be found, or if \pausing is nonzero. \errorstopmode is the default. If you aren't in \errorstopmode, your .log file will contain "help" messages for all of your errors. These modes are global (they don't revert at end of group). They can be set in a format file; thus you can have a format that implies batch processing. But you could override that, e.g. by running "batchtex \errorstopmode \input paper". (This example assumes that batchtex is a program representing "virtex&batch", i.e., a virgin TEX with batch.fmt loaded and then the core image saved.)

(40) \dump will save TEX's current memory contents. \dump is essentially like \end (it's the last thing you do with INITEX), and you don't specify a file name. If your input was named foo, your output file will be named foo.fmt. Such files are now called format files. This is allowed in INITEX only, and only at very quiet times (i.e., at group level 0 in vertical mode with nothing on the current page, etc.). The file name will be printed later when these memory contents are loaded in a production version of TEX; for example, if you say "\dump" on March 1, 1984, the TEX that uses the dumped file might begin with the line

```
This is TeX, Version 1 (format=plain 84.3.1)
```

(41) Actually the program name TEX now stands only for versions of TEX82 that have PLAIN.TEX preloaded. Other preloaded versions (e.g. AMS-TEX) will usually exist too. If your operating system does not allow a program to start with its memory preloaded, you will have to call a "virgin TEX" program VIRTEX that first wants to see the name of a format-dump file (e.g., PLAIN or AMSTEX). In this case a typical calling sequence might be "@virtex &amstex paper". If no format is given, "&plain" is assumed. If

your operating system is nice enough to allow preloaded programs, a typical way to create the program TEX would be to say "`&virtex &plain`" followed by something like "control-C" and "`save tex`".

(42) `\indent` takes you from vertical mode to horizontal mode and indents the paragraph; this can be used if the first item in the paragraph is in an `\hbox` or `\vbox`. You can also use `\indent` in horizontal mode to stand for "`\hbox to\the\parindent{}`".

(43) If you end the parameter part of a definition with an additional `#`, the argument-matching process will terminate on the next left brace. For example, in

```
\def\chop to #1#{\chopp{#1}}
```

the call "`\chop to 2in{x}`" will expand to "`\chopp{2in}{x}`". The definition

```
\def\mac#{why}
```

will subsequently issue an error message if "`\mac`" is not followed by "{".

(44) The "`texinfo`" that is given with each font (see Appendix F of the **METAFONT** manual) can now be changed by a TEX user program; there was no way to do this before except by making a new TFM file. Say

```
\texinfo ⟨font⟩⟨parameternumber⟩=⟨dimen⟩
```

For example, `\texinfo\ff3=4pt` sets the stretch component of spacing to `4pt` in font `\ff`. (Parameter 1, the "slant", is unitless but you should give its value in units of points.) You can use this feature to adjust math-mode positioning of subscripts, etc., by changing the parameters in **mathsy** and **mathex** fonts. Note that `texinfo` is global, it does not get reset at the end of a group. You can also say `\the\texinfo⟨font⟩⟨parameternumber⟩`.

(45) New dimension parameters `\hfuzz` and `\vfuzz` specify the tolerance for printing a diagnostic message about overfull boxes. If the box is overfull by this amount or less, no message is printed. Default is "`.1pt`", which was the old TEX standard. If you say `\hfuzz 8000pt`, you probably won't see any overfull boxes. If you say `\hfuzz 0pt`, you will see all of them, including a few that you didn't know about last year.

(46) Another new dimension parameter `\overfullrule` specifies the width of a rule that is added at the right end of overfull `hboxes`. This rule has the height and depth of the box. If `\overfullrule` is zero or negative, or if the amount of overfullness does not exceed `\hfuzz`, no rule will appear. Default is `5pt`, which gives a big black mark to help you spot overfull boxes.

(47) The "`overfull box`" warning messages will be given in a new form that simply gives the characters in the box; for example,

```
Overfull box, 3.3326 points too wide (in paragraph of lines 210--216):
\ff This is the text of a line that was over-full for some rea-son.
```

Discretionary hyphens are shown as real hyphens, so that you can see what hyphenation TEX was trying. The error-transcript file gets both this message and an old-style description of the overfull box in detailed diagnostic dump format.

(48) Another new control sequence, `\relax`, does nothing at all. Thus, if you want to disable the action of a control sequence, you can `\let` it be `\relax`.

(49) Up to 256 fonts may be used, and each font may contain up to 256 characters. (Characters numbered 128 to 255 can be accessed either via ligatures or charlists or with the `\char` command.)

(50) `\leftskip` and `\rightskip` specify glue to be placed at the left and right of each line of a paragraph. This provides better ways to do ragged right setting, and it makes changes to `\hsize` less necessary.

(51) `\lastskip` gives the value of the previous item in the current horizontal or vertical list, if that item was glue, otherwise it yields the value `0pt`. Thus, to get the effect of `\unskip` in vertical mode, say "`\penalty100000\vskip\minusthe\lastskip`". You can also do things like "`\ifdim\the\lastskip > 5pt...\else...\fi`" so that one macro can make decisions about spacing based on what has gone before. (This finally solves the long-standing problem about spacing after theorems that end with a displayed equation.) Note that `\the\lastskip` is permitted, except in `\write` statements.

(52) `\sqrt` signs in TEX82 are positioned differently in their boxes: The baseline now comes exactly at the bottom of the place where the vinculum (i.e., the rule over the operand of `\sqrt`) is to be joined. This

means that no rounding errors will be possible and perfect alignment will be obtained at all resolutions. Pre-82 versions of TEX will still work (subject to rounding) if the height of the box is the thickness of the rule.

(53) The error transcript files are no longer called "errors.tmp". Your output file and transcript file will be "paper.dvi" and "paper.log" if your first line of TEX input specifies \input paper. The default name "texput" is used whenever no other appropriate name has occurred before TEX reads line two of its input. (TEX can't wait any longer, since line one has to be put into the transcript file, and the transcript file has to have a name before it gets information.)

(54) \hyphenation{word list} can be used to override TEX's hyphenation algorithm; for example, to specify hyphenation of the words "hyphenation" and "exceptions" one can write

```
\hyphenation{hy-phen-a-tion ex-cep-tions}
```

A new hyphenation algorithm devised by Frank Liang is used in TEX82; this one extends much more readily to other languages. Words containing ligatures can now be hyphenated automatically, even difficult words like "difficult".

(55) If two fonts are specified with the same name and point size, only one will be loaded.

(56) \sfcode ⟨char number⟩ is the spacefactor code for that character, times 1000. For example, the spacefactor code for period and question mark is normally 3000, for comma 1250, for right parenthesis 0 (meaning do not change the space factor), and for most characters it is 1000. In TEX82, you also say "\spacefactor 1234" instead of "\spacefactor 1.234".

(57) TEX82 has a new "help" facility available on error messages. If you type "h" after an error, you will (usually) get further explanation of what the error means, together with suggestions about how to proceed.

(58) \↑↑\ and \↑↑] have gone away, to the delight of people who don't have nice ways to type ascii control characters. Instead, \nonscript in math mode precedes a space of any other type, making that space zero in subscript styles. Thus, the conditional thin space is now "\nonscript\mskip\the\thinmuskip", and conditional negative thin space is "\nonscript\mskip\minusthe\thinmuskip".

(59) \thinmuskip, \medmuskip, \thickmuskip are now definable like other glue parameters such as \baselineskip. The units should be in mu. For example, one of the defaults is \thickmuskip 5mu plus 5mu. The old "mathspace" parameter in symbol fonts (see **METAFONT** manual p. 99) is no longer used.

(60) There's a new way to get up to 4096 more math symbols in all three sizes, by defining font families 0 to 15. For example, suppose that fonts \fA, \fD, and \fF are Fraktur alphabets in 10pt, 7pt, and 5pt sizes. Then you can say

```
\textfont 5=\fA \scriptfont 5=\fD \scriptscriptfont 5=\fF
```

which is something like the code "\mathrm adf" in the old basic.tex. Now if you say "\fam5" in math mode, you get characters from font \fA, \fD, or \fF, depending on the size. For example, "{\fam5 B↓b}" would give Fraktur B in 10pt with a subscript Fraktur b in 7pt. The rule is that a family specification overrides the normal family for symbol of class var (see below).

Note that we can now say \def\rm{\fam0\ff} together with

```
\textfont 0=\ff \scriptfont 0=\fg \scriptscriptfont 0=\fh
```

and then it's possible to say, e.g., "\def\max{\mathop{\rm max}}" instead of resorting to "{\char`m \char`a \char`x}" in order to achieve size-switching. This extension also makes ligatures and kerning available in math mode.

(61) The use of certain families is predefined. Family 2 specifies the 'mathsy' fonts used for symbols; family 3 specifies the 'mathex' fonts used for large delimiters; and the other 14 families can be used in any desired fashion. Instead of '\mathsy uxz' one now says

```
\textfont 2=\fu \scriptfont 2=\fx \scriptscriptfont 2=\fz
```

and these assignments are local (they go away at the end of a containing group). You must have \textfont2, \scriptfont2, \scriptscriptfont2, \textfont3, \scriptfont3, and \scriptscriptfont3 defined before using math mode, since the parameters of these fonts contain the

values TEX needs for math spacing. (\scriptfont3 and \scriptscriptfont3 are now supposed to be math extension fonts, as well as \textfont3, because TEX82 will use smaller extension-type features in script and scriptscript styles; for example, the default rule thickness in a subscript is a parameter to \scriptfont3, while TEX80 had only one extension font for all three sizes.)

(62) The mathcode now has a 15-bit number as its value. The first three bits specify ord, op, bin, rel, open, close, punct, and var (where var is like ord but it substitutes the "current" family for the stated one). The other twelve bits specify a "math character", with four bits for the family and eight for the character. For example, character ´*100* in family 5 is ´*2500*; of course, this reads a little better in hexadecimal: character "40 in family 5 is "540.

You can say \mathchar followed by such a 15-bit code, to get the equivalent of typing a character with that math code. Thus, one can now say, e.g.,

```
\def\cdot{\mathchar ´10001 }
```

instead of \def\cdot{\mathop{\char1}} as formerly. The control sequence \char is no longer allowed to take values greater than 255.

(63) If the first three bits of a mathcode are 7 (case "var"), it's the same as 0 except that the current value of integer parameter \fam replaces the family specification (if it is between 0 and 15). The current family is set to −1 whenever math mode is entered.

(64) A \mathcode can also be ´*100000*; this causes the corresponding character to behave as if it had catcode 13 (if it is a letter or otherchar). For example, plain TEX defines the mathcode of ´ to be ´*100000*, and control sequence ´ is defined to be ↑\prime, so that you can say f´(x) instead of f↑\prime(x). In this way you can have characters that are active only in math mode. (Furthermore, an octal constant like ´*77777* will still be acceptable in math mode.)

(65) New primitive \mathchardef will save lots of memory if you have lots of control sequences for math things like \alpha, etc.:

```
\mathchardef\cs[=]⟨fifteen bit number⟩
```

will have the effect of

```
\def\cs{\mathchar ⟨fifteen bit number⟩ }
```

and it will use none of TEX's memsize. (By contrast,

```
\def\cs{\mathchar "1234 }
```

takes up 9 words of memory!)

(66) The \radsign parameter goes away. Instead, one says \radical followed by a delimiter code. Delimiter codes may be used also after the control sequence '\delimiter' in connection with \left and \right and \atopwithdelims.

A delimiter code is a somewhat esoteric 24-bit number. The first twelve bits specify a 'small' character, and the last twelve bits specify a 'large' one. When TEX chooses a delimiter, it searches in the following way until finding the first one large enough: First it looks at the 'small' character in the current size of the family, then (if the current size isn't text size) it looks at the small character in the next larger size, and so on until coming to text size. If a suitable delimiter has still not been found, the same search is carried out starting at the 'large' character. If any of the characters looked at is part of a "charlist", the list is searched before moving on. If the small or large character is zero, it is ignored; thus, you can't use character 0 in family 0 as a delimiter.

For example, \sqrt is equivalent to \radical ´11601560 in the Computer Modern fonts; the *1160* specifies \fam2\char´160, and the *1560* specifies \fam3\char´160. Since ´*160*="70, we can also write this as \radical "270370.

If \delimiter ⟨delimiter code⟩ appears in a formula in some place not controlled by \left or \right or \atopwithdelims, it is actually a 27-bit code. The least significant 12 bits are ignored, and the leading 15 bits are used like a \mathchar (thus, they specify a category as well as a family and character). The reason is that one can now say, e.g.,

```
\def\lfloor{\delimiter ´411421404 }
```

so that one can say both \left\lfloor and simply \lfloor.

A \delcode is also given for letters; \left and \right and \atopwithdelims will use this code as a delimiter code if they are followed by an "otherchar". For example, the Computer Modern fonts use \delcode`(='501400, assuming that family 0 contains the ordinary roman alphabets. Initially, \delcode is negative for all characters; this denotes an invalid delimiter.

All this bit hackery is, of course, unfriendly looking, but the goal is to make it possible for macro packages to define the friendly codes without taking up much memory space inside of TEX. All of the TEX control sequences that used to be predefined for Computer Modern are now unbundled so that arbitrary encodings can be used. One of the embarrassing limitations of TEX80 was that it could handle delimiters only between *'142* and *'153* in the symbols font, and it insisted that these 'small' delimiters had corresponding 'large' ones in positions *'004–'015* of the mathex font!

(67) The spacing in math formulas is unbundled too; there are three parameters \thinmuskip, \medmuskip, \thickmuskip to specify the spacing in formulas like $x\log x$, $x+x$, and $x=x$, respectively. One defines these using "mu" units, e.g. '\medmuskip = 4mu plus 2mu'. The control sequences '\,' '\>' and '\;' in math mode yield spaces of these three varieties. (These are now defined in PLAIN, not TEX primitives. The previous meaning of \> is now rendered '\nonscript\>'.)

The following example shows a feature that is *not* allowed:

```
$a \save1\hbox to 18mu{}
        \ifdimen 1wd1=10pt{\gdef\x{\over b}} \else{\def\x{}}
        \x$
```

Whoever wrote this was trying to be clever and discover whether TEX was in \textstyle. But the program is self-contradictory, because if the "a" is in 10pt text style the formula changes itself to $a\over b$ where the "a" is in script style, so the formula changes itself to a where the "a" is in text style, so .... Constructions like this show why TEX does not allow variable dimensions like mu except in very restricted ways like \mskip.

(68) New dimension parameters

\scriptspace (this amount is placed at the right of all subscripts and superscripts; TEX80 used about .45pt always)

\nulldelimiterspace (this amount is used before and after all fractions defined by \atop, \over, \above, and for all "." delimiters)

\delimitershortfall (see the next parameter)

(69) New integer parameter \delimiterfactor. When TEX computes the size of \left and \right delimiters, it computes $delta_1$ = twice the maximum distance of the enclosed formula from the "axis". (The axis is where the fraction line would go.) Let $delta_2 = delta_1 * (\texttt{delimiterfactor}/1000)$ and $delta_3 = delta_1 - \texttt{delimitershortfall}$. The delimiters will be as small as possible provided that their height+depth exceeds both $delta_2$ and $delta_3$. (TEX80 took delimiterfactor=900 and delimitershortfall=1ex; TEX82 lets the user twiddle with these magic numbers.)

(70) When unscripted letters occur in math mode, they now are adjusted for ligatures and kerns. This means, for example, that "*df*" will be spaced better, once appropriate kerning information has been added to the math italic fonts. The (new) rules for spacing are this: If there's no kerning specified, add the italic correction to every symbol in math mode; otherwise use the kern (without the italic correction). However, a subscript is moved left by the amount of italic correction. In formulas like $P↓2↑2$, the 2's will no longer be directly above each other (the sub-2 will be to the left of the sup-2 by the amount of the italic correction). TEX80 put them above each other, thereby following a long-standing convention (cf. Oxford book by Chaundy et al.), but this usually turned out to be undesirable, so people started to write $P↑2↓{\!2}$ all the time. If anybody really wants it the old way, they can get it by $P{}↑2↓2$.

(71) "\comb" is changed to "\atopwithdelims". There's also "\overwithdelims" and "\abovewithdelims". For example, {1\overwithdelims[]2} is sort of like \left[1\over2\right]. And "\abovewithdelims..." is equivalent to "\above".

(72) The maximum penalty has been raised from 1000 to 10000; thus, "\penalty 1000" no longer absolutely prohibits a break, but "\penalty 10000" does. This number 10000 is being used elsewhere in TEX82

also: For example, \tolerance or \pretolerance of 10000 is equivalent to saying "use all possible line breaks, regardless of how much stretching is necessary." A \penalty of -10000 (or less) is equivalent to the old \eject in vertical mode, and to the old \linebreak in horizontal mode. These are no longer primitives of TEX82.

(73) The \pagebreak feature is also eliminated, in favor of a much more general feature. You can say \vadjust{vertical list} in the midst of any paragraph, and whatever is in the specified vertical list will be placed immediately following the box for its line when the paragraph has been made. For example, \pagebreak is now written \vadjust{\penalty-10000}. You can use this feature to do things like insert extra space between lines of a paragraph, something like \noalign does for alignments.

(74) \ifvoid n tests if \box n is not present. All boxes are initially void; a null box (made e.g. by \hbox{}) is empty but present. A box becomes void after it is used.

(75) \topbaseline has changed to \topskip; thus glue is allowed at the top of a page. This makes it easy to "bottom justify", for example. (The old \topskip and \botskip are no longer used.)

(76) \vtop is now allowed in any mode, just like \vbox.

(77) \special{keyword arg} is a general extension feature. The keywords are system dependent, but TEX copies "keyword arg" into the DVI file so that any device driver that knows your keywords will do the right thing with them. Users should get together if they want to standardize on various keywords. Examples: \special{halftone fig22} could mean "insert a halftone from file fig22, with its reference point at the current reference point"; \special{leftend 2} and a later appearance of \special{rightend 2} could mean "draw a straight line from the left reference point to the right one" (the "2" is an identifier to distinguish this line from another one); \special{message Foo} could mean "display 'Foo' on the console of the printing device"; and so on. Semantically, \special acts like a box of height, width, and depth zero, as far as TEX is concerned; the argument in braces is sent to the DVI file where it is associated with the current reference point. The length of "keyword arg" must be at most 255 characters.

(78) \ragged is no longer implemented, since \rightskip does ragged right setting so much better.

(79) \ifcase ⟨number⟩ case0 \or case1 \or case2 \else remaining cases \fi illustrates a new way to choose between more than two alternatives without too many nested brackets. Any number of cases can be given (but TEX has to scan by them each time, so it's best to define them as control sequences if they are long). The \else is optional.

(80) Major changes have been made to the \output conventions, so that it will be easier to produce balanced columns and various other things. The old ideas of \topinsert, \botinsert, \topsep, \botsep are eliminated!

In their place one says \insert n, where n is a box number, e.g. "\insert 250". Different numbers correspond to different classes of insertions; for example, one might want to have figures as well as footnotes inserted at the bottom of pages, and TEX80 used to use the same treatment for both. Under the new conventions, all \insert 250's that go on a page will appear in \box 250 when the \output routine starts.

The old idea of "\page" is gone too, and the meaning of \output has changed, so read carefully: The contents of an accumulated page, exclusive of inserts, is placed into \box 255, so that the output routine can place this material together in whatever way it wants. (\insert 255 is not legal.) A new TEX primitive called \shipout, followed by a box specification (e.g., "\shipout\vbox{\box255\box250}") is what actually produces output. Note that \ifvoid 250 can be used to test whether any \insert 250's have been gathered for a page. The \shipout command can be used anywhere, not just in \output. Incidentally, \shipout prevents the old anomalies about the values of \counts being different from what people thought they would be when \writing table-of-contents or index data to a file. The default \output routine in TEX82, if none is specified, is "\output{\shipout\box255}".

Another new primitive, \vsplit, is handy for multi-column input. The command "\setbox 2=\vsplit 250 to 100pt" will, for example, make \box2 a box whose height is 100pt, by extracting 100pt worth of material out of \box250. The depth of \box2 will be at most \splitmaxdepth (using the rules that TEX used for \page); \vsplit extracts the optimum initial segment of \box250, in the sense that

badness + penalty is minimized and the segment is as long as possible subject to this condition. After \vsplit has acted, \box250 will contain the residual, as if a page break had occurred; thus, glue and penalties will be eliminated following the break, and the first box or rule (if any) will then be preceded by sufficient glue to position its top baseline (based on \splittopskip).

The main idea of \vsplit is to make it easier for an \output routine to produce multiple column format. For example, if \box255 is 300pt high, one can get triple columns by

```
\setbox 1=\vsplit 255 to 100pt
\setbox 2=\vsplit 255 to 100pt
\setbox 3=\vsplit 255 to 100pt
```

(Actually it is safer to use slightly less than 100pt here, but the exact measurements depend on the top baseline and other things.) The remaining material in \box 255, if any, can be put back onto the following page, as we will see momentarily; and boxes 1,2,3 can be positioned as desired before they are shipped out.

The \vsplit routine also looks at \marks in the contained box. It sets \splitfirstmark and \splitbotmark to the topmost and bottommost contained marks; otherwise it sets these to null strings.

Of course you can't apply \vsplit to a box that was constructed as an \hbox. There is no \hsplit.

The old \output routine defined a sequence of items in restricted vertical mode, with the meaning that this sequence would be vboxed and shipped out. The new \output routine defines a sequence of items in restricted vertical mode, with the meaning that this sequence will be placed in front of whatever TEX has accumulated for the following page, including whatever caused a break on the current page. Thus, if you write "\output{\unvbox255}" you are in serious danger of getting in an infinite loop.

Consider, for example, what happens if a page break occurs at some glue. If the output routine leaves some of the tail end of the material from \box 255 in its vertical list, this material will fit perfectly before the glue that caused the previous break, since the glue following a break is not eliminated when \box255 was made; glue is simply discarded when it appears at the top of the new page that is started after \output finishes.

Consider also what happens if a page break occurs at "\penalty-10000". If you understand what has just been said, you will see that this would cause repeated looping if the \output routine places something back for the next page, and this might be a problem. Therefore TEX will change the penalty to +10000 at a break, and it also sets \outputpenalty to the value of the penalty that actually caused the break. (\outputpenalty is set to 10000 if it was glue that caused the break.) Thus, you can restore the effect of a penalty by putting "\penalty\outputpenalty" at the end of your output routine.

But how does TEX figure out what to put in \box 255 and how much of the insertions to put into other boxes, before it calls on your output routine? The rules are slightly complicated, but they have been devised to handle a wide variety of situations, including situations where some inserts span several columns. It should be possible to do things like put footnotes in two columns beneath single-column text, or to put single-column footnotes beneath double-column text, etc. Here's how: We associate \skip n, \dimen n, and \count n and \box n with \insert n. (a) \count n gives a magnification ratio of \insert n with respect to ordinary text. For example, if two-column footnotes go with one-column text, and if footnotes are inserted with \insert250, then \count250 should be 500. If single-column footnotes or page-wide figures are being inserted with double-column text, the magnification ratio should be 2000. (b) \dimen n gives a maximum length of inserts for box n; subsequent inserts will be carried over to a following page. (c) \skip n gives a correction term when there is at least one insertion for box n.

The total length of inserts is figured as the sum, over all n such that \insert n appears on a page, of the \skip n plus (\count n over 1000) times the total natural height plus depth of all \insert n's, including the original contents of box n before any inserts were made. The badness of a page is computed from the amount of the text on the page plus the total length of inserts, and TEX breaks

the page so that badness is minimized. The \vsize is the total of text plus insertions; if you are using double-column text, your \vsize should be about twice the actual page height.

If this isn't complicated enough, there is also a rule for splitting inserts, so that long footnotes can be broken between pages and so that large figures can be carried over to subsequent pages. Here's the idea: When we are deciding whether to perform an \insert n or not, we first look to see if previous \insert n's have all been completed without splitting. If not, this one is carried over to the next page. If so, this one is put on the current page, if it does not cause the page to overflow and if it does not cause the maximum (\dimen n) to be exceeded. In the latter cases, the insertion is \vsplit to the maximum size that would not cause such overflow. For example, suppose we get to an insertion at magnification 500 when \vsize minus the current amount of text and the previous total amount of insertions leaves only 50pt of vertical space left. Suppose the insertion takes 150pt of vertical space, so that it would take 75pt after scaling; and suppose that 150pt would not exceed the maximum total size of insertions for this box. Then we essentially \vsplit the insertion list to 100pt (this will scale down to 50pt), after which the actual length of the insertion will be computed as its natural height plus depth (which might be different from 100pt). The remaining part of the insertion will be corrected for top baseline, etc., as in an ordinary \vsplit, if this broken insertion is actually chosen. But TEX will use broken insertions only if they lead to the minimum badness for the resulting page.

If several \insert n's appear on the same page they are concatenated together with no baselineskip correction between them. So you should use struts to produce the correct positioning.

Insertions are treated the same whether they appear in horizontal or vertical mode. A "floating" insertion turns out to be a special case of a broken insertion, whose first component is a null box.

TEX doesn't put waiting insertions into \box255, it leaves them on the list for the subsequent page. If insertions appear in a box that is being vsplit, they are ignored.

It's too bad that these rules came out so complicated, but in simple cases the output routines will now be quite simple, and the manual will have enough examples to make things clear (I hope). Nothing simpler than this seems to provide the other features that people have been demanding, and the total amount of programming for the TEX82 page builder is not much more than there was in TEX80.

(81) In vertical mode, you can say \prevdepth=3pt to make TEX82 act as if the previous box had a depth of 3pt, when computing the glue between boxes to achieve the baselineskip. If you set \prevdepth to a value less than or equal to -1000pt, the baselineskip calculation will not be made. (This is the case at the beginning of a vbox, or just following an hrule.)

(82) You can say \the\spacefactor and \the\prevdepth, if you are in horizontal or vertical mode, respectively.

(83) New diagnostic features: "\showbox 10" will display the current contents of \box10 on the terminal. There's also \showthe as in "\showthe\count 5" and "\showthe\baselineskip". Also "\show\cs" to give a symbolic display of the current meaning of the control sequence \cs. The present \ddt is deleted, and \showlists exhibits the current activities the way \ddt used to. If you are trying to diagnose some mysterious behavior, you can say, for example, "\showthe\texinfo\ff 40" and you will get an error message like "\font \ff has 7 texinfo parameters" (if it has fewer than 40). Incidentally, if your error message was "\font \fg has 7 texinfo parameters", you would know that fonts \ff and \fg are being treated identically. (TEX loads only one copy of a font that you mention twice. If you want to load two distinct copies, so that you can diddle their parameters independently, you can try something like this: "\font \ff=cmr10 at 10pt \font \fg=cmr10 at 10.00002pt".)

(84) If you use a \skip parameter in the context of a dimension, the natural width is used. For example, \setdimen 5=\the\baselineskip. If you use a \dimen parameter in the context of an integer, the conversion is in units of sp (scaled points, 2^{-16} of a pt). For example, \setcount 10=1truept would set \count 10 equal to 65536000 divided by \mag.

(85) When you use \advskip, infinite glue wipes out finite glue.
For example, "\setskip 2=5pt plus 2pt minus 1fill
\advskip 2 by 3pt plus 1fil minus 1fil"
is equivalent to "\setskip 2=8pt plus 1fil minus 1fill".

(86) `\linepenalty` is yet another parameter to control line breaking. TEX82 adds this to the badness before squaring to get demerits. (Previously, I had `\linepenalty=1` always; by setting it a bit higher, like maybe 7 or 8, you tend to get paragraphs that are set tighter when a line can be saved. I don't think it's a good idea to make `\linepenalty` real large, and it would be foolhardy but weird to make `\linepenalty = -10000`; this apparently would minimize the number of lines but maximize the badness!)

(87) Macro parameters are now delimited by strings instead of single items. For example, `\def\a#1ab{...}` followed by `\a acaab` will set `#1` to "aca"; `\def\a b#12#{foo#1}` followed by `\a bbar2baz2{8}` will expand to `foobar2baz{8}`; the latter followed by `\a 2...` will give an error message (`\a not followed by b`). You get the error message only if there's a string before the first parameter.

(88) Characters catcoded 13 are not equivalent to one-letter control sequences. They act like control sequences (e.g., you can use them after `\def` and `\let`), but `&` and `\&` will be distinct.

(89) `\everypar{...}` inserts its argument into TEX's scanner at the moment TEX has changed from vertical to horizontal mode. The paragraph indentation will already appear in the paragraph, unless of course the transition to horizontal mode was due to `\noindent`.

(90) Spanned and omitted columns in alignments: If an entry in an alignment is '`\omit`', the preamble text for the column is omitted in this row. The control sequence `\span` can be used in place of a tab mark, and the result is that the surrounding entries are combined together. You can use `\omit` only as the first item of a column. If you use `\cr` in place of a tab mark, the effect is as if all further columns in that row are omitted; thus, your preamble can specify more columns than are actually used.

Example:
```
\tabskip 1em plus 1em
\halign to 20em{\ctr{#}&\rt{#}&\lft{#}\cr
                AAA&B&C\cr
                DDDDDDD\span\omit&EE\cr
                \omit\span FFFFF\span\omit\cr
                G&\omit H\span III\cr}
```

where all the letters are 1em wide, say. Let w_{ij} be the maximum width of the entries that span columns i thru j. The first line "`AAA&B&C`" implies that $w_{11} \geq 3$, $w_{22} \geq 1$, $w_{33} \geq 1$. The second line says that $w_{12} \geq 7$ and $w_{33} \geq 2$, and so on; we find that $w_{11} = 3$, $w_{12} = 7$, $w_{13} = 5$, $w_{22} = 1$, $w_{23} = 4$, $w_{33} = 2$. Column widths are now assigned from left to right, as follows:

$$c_1 = w_{11}$$
$$c_2 = \max(w_{22}, w_{12} - t_1 - c_1)$$
$$c_3 = \max(w_{33}, w_{23} - t_2 - c_2, w_{13} - t_1 - c_1 - t_2 - c_2)$$

where t_i is the natural width of `tabskip` between columns i and $i+1$. In this case $t_1 = t_2 = 1$, so $c_1 = 3$, $c_2 = 3$, $c_3 = 2$. This means the natural width of the lines will be $1+3+1+3+1+2+1 = 12$ ems, so the glue will be stretching to make up the additional 8ems; each unit of stretch is doubled. When columns are spanned, however, TEX justifies the material into a box having the appropriate width for the `tabskip` glue that was omitted; for example, an entry that spans columns 1 and 2 will be justified to width $c_1 + t_1 + s_f * s_1 + c_2$, where s_1 is the stretchability between columns 1 and 2, and $s_f = 2$ in this case since the glue is being doubly stretched. (If the `tabskip` glue shrinks, we would of course use shrinkability instead; in this case spanned columns might actually get smaller than their natural size.)

The result of the above example, taking account of which parts of the preamble are omitted by the `\omit` operations, is therefore

```
|  AAA     B   C    |
|   DDDDDDD    EE   |  (columns 1 and 2 spanned and centered)
|          FFFFF    |  (columns 1 to 3 spanned, flush right)
|   G   HIII        |  (columns 2 and 3 spanned, flush left)
```

Restriction: A single entry can span at most 256 columns.

(91) Spaces are ignored after tab marks in alignments.

(92) If every entry in an alignment that uses column j also spans column $j+1$, the `tabskip` glue between columns j and $j+1$ is effectively eliminated. Similarly, if column j never appears (because each row had `\cr` before getting to that column), the `tabskip` glue after it is eliminated.

(93) \hbox par is eliminated! Instead of "\hbox par 100pt{...}", one now says "\vbox{\hsize 100pt ...}" and the effect is almost the same. The only difference is that the paragraph or paragraphs in the \vbox will have both their hanging indentation and looseness (and perhaps also their baselineskip) specified inside the vbox; this is, of course, more logical than the old rule.

Restricted vertical mode is no longer restricted; it's called "internal vertical mode". It differs from vertical mode only in not going through the page builder, and in allowing \unskip as well as \vskip with infinite shrinkability.

(94) To have the paragrapher work on inserted text (e.g., in footnotes), one writes "\insert 250{\hsize 200pt ...}", perhaps using the \interlinepenalty parameter that adds to the penalty between lines of a paragraph (whether in inserts or not).

(95) Three new dimension parameters \displaywidth, \displayindent, and \predisplaysize are assigned values at the beginning of every displayed formula: (1) \displaywidth is the length of the line that will contain the formula before it is centered; this is usually equal to \hsize, except when hanging indentation or \parshape are being employed in a paragraph. (2) \displayindent is the amount by which that line is indented. (3) \predisplaysize is the amount of copy on the line preceding the displayed formula; this is what is used to decide between \dispskip or \dispaskip (*these names change; see #142*). If the display immediately follows \noindent or another display, \predisplaysize will be $-(2^{30}-1)$ sp (the smallest legal dimension in TEX). Otherwise, if the position of the last box on the previous line is affected by glue stretching or shrinking, \predisplaysize is set to $+(2^{30}-1)$ sp. Otherwise \predisplaysize is set to the natural width that the preceding line would have if all glue were removed at its right end, plus the amount of indentation of that line, plus 2ems.

(96) \setcount, \setdimen, \setskip, \setbox, \output, and \everypar are local definitions unless specified global. Likewise the results of \advcount, etc.

(97) The { } in \output still defines grouping (it would be too dangerous to leave it out, since \output occurs asynchronously), but the { } in \everypar does not. Grouping is now independent of \if tests, as explained later (*#120*).

(98) All of TEX's primitives now mean the same thing in all modes. The control sequences that were exceptions to this rule have been dealt with as follows:

\␣ (control space) now means a text space, even in math mode.

\quad is no longer a primitive (PLAIN defines it as \hskip 1em).

\! is no longer a primitive (PLAIN defines it for math mode only)

and \ignorespaces is a new primitive that gobbles spaces.

\- is always a discretionary hyphen; its previously advertised mathmode function has been taken over by the new \nonscript primitive.

(99) \accent is allowed only in horizontal mode; \mathaccent only in math mode. The latter takes a 15-bit math code, the former an 8-bit character code; it's like the difference between \char and \mathchar.

The \mathaccent primitive will make use of a charlist of characters to choose the first accent of a list whose successor is either nonexistent or wider than the formula being accented. Thus, you can have a list of longer and longer tildes or hats, etc.

(100) Spacing in math is slightly different: Fractions and \left...\right subformulas are given type Inner, so there are eight types instead of seven. (Previously Inner was treated like Ord.) The spacing matrix entries are set so that there's at least a thin space between Inner and its neighbors, except in the pairs Open Inner and Inner Close. There's a new primitive \mathinner analogous to \mathpunct, etc.

(101) "\vcenter to 100pt" is now allowed in math mode, if anybody wants it.

(102) "\ifdimen" is renamed "\ifdim" and there's also "\ifnum" replacing "\ifpos".
Examples: \ifnum\count1>5...\else...\fi; \ifdim\dimen3<1.5wd2...\else...\fi.

(103) "\ifinner" is true if the current mode is internal-vertical, restricted-horizontal, or non-display-math. Combining this with \ifvmode, \ifhmode, \ifmmode makes it possible to determine exactly what mode you are in.

(104) New primitive \kern allows you to specify unbreakable space (without stretching or shrinking). Thus, "\kern -1pt" is something like "\penalty10000\hskip-1pt". There's also "\mkern 3mu" in math mode. You are allowed to use \kern but not \hskip in \discretionary lists. A \kern in a word does not upset the hyphenation algorithm. You can use \kern in vertical as well as horizontal lists. It is legal to break at a kern if it is immediately followed by glue or leaders, provided that it is not preceded by glue, kern, or penalty.

(105) New debugging facility \tracingcommands, if nonzero, gives a symbolic indication of what commands are being obeyed by TEX's main control routine.

(106) New integer parameter \maxdeadcycles gives an upper limit on how many consecutive invocations of \output do not cause at least one \shipout. This is intended to catch unintended loops. Default is 25.

(107) If \tracingstats > 0, you get to see how close you came to TEX's current table capacities, in a list of statistics printed at the end of your run.
If \tracingstats > 1, you also get to see the current memory usage every time you do a \shipout.

(108) If \tracingparagraphs > 0, you also get a huge amount of inscrutable printout about what the line-breaking algorithm thinks it is doing.

(109) If \tracingpages > 0, you get similarly inscrutable printout about what the page-breaking algorithm thinks it is doing.

(110) But \tracingstats, \tracingparagraphs, and \tracingpages are ignored unless TEX has been compiled in a "slow version" that actually maintains these statistics. (The SAIL version of TEX is currently "slow" in this way.)

(111) \string replaces the next token by its text, with all characters regarded as type otherchar (except that a space will be of type spacer; it's possible but not easy to get a space here). For example, \string\abc results in four characters \, a, b, c. This expansion occurs just as for \number, i.e., when \string occurs in \xdefs or in horizontal or math mode. (The most common use of \string is to follow it with a macro parameter.) Caution: If characters in a control sequence name are nonstandard in ascii, they will be converted differently at different installations.

(112) Here is an extension to the language intended to placate people who have objected to the fact that \write (and \openout and \closeout) only cause action at the time of the next \shipout. Some applications call for immediate output, hence a new feature: \immediate followed by \openout or \write or \closeout causes the output action to take place without delay. For example, \immediate\write0{x} is equivalent to \shipout\vbox{\write0{x}} except that the latter also puts an empty page into the DVI file.

(113) New parameter \boxmaxdepth affects \vbox: If the depth of the box would exceed \boxmaxdepth according to the normal rules, the box contents are shifted up so that the depth is exactly \boxmaxdepth, before setting the glue. The same applies to \vtop, before adjusting its depth. The default setting is \boxmaxdepth='7777777777sp (the maximum dimension).

(114) You can now use \let with non-control-sequences after the = sign. For example, \let\zero=0 makes the control sequence \zero behave something like the digit zero; but if you want to make the constant 100 by saying "1\zero 0" you still have to \def\zero{0}. Thus, this new extension isn't a big breakthrough, but it does save a bit of space and time inside TEX. (Incidentally, after \let\zero=0, \zero will not expand to 0 in xdefs.)

(115) Popular demand wins again:
You can now say \csname ⟨string⟩\endcsname to manufacture a control sequence name. For example, \csname foo \endcsname is essentially identical to a control sequence named "\foo " (note that the space is part of that name!) and \csname foo\endcsname is like "\foo" and, after \def\zero{0}\def\test{zero}, it follows that \csname\csname\test\endcsname\endcsname is like "\0". The conversion from token list to control sequence occurs as if \csname were a macro being expanded. If the control sequence hasn't been defined before, it will behave as if it were "\relax".

(116) In the preamble to \halign or \valign, the primitive \span would normally make no sense. But it causes TEX to expand the following token, instead of just copying it, before inserting that token in

the preamble. (Previously this was possible only with a dirty "\tabskip" trick, since TEX expands whatever follows "\tabskip 0pt" looking for "plus".)

(117) Syntactic conditionals!
For years people have been asking for TEX to treat conditionals in its "mouth" rather than in its "stomach", and I have been fending them off. But starting with Version 0.8 of TEX82, \if tests are made at the time of macro expansion rather than as part of the semantic processing in horizontal or vertical or math mode.

Instead of writing \if...{..a..}\else{..b..} the new syntax is

\if... ..a..\else..b..\fi

(with \else optional if ..b.. is empty).

Whenever TEX is reading material in a mode where macros are now expanded, it will process conditionals somewhat as though they were macros. Namely, \if... results in evaluating the condition and skipping code if the condition isn't true (skipping to the next \else that isn't enclosed by \if..\fi brackets); \else, \or, and \fi switch in the appropriate way between reading and not reading text.

(118) Braces need not be properly nested inside the conditionals, nor do \if...\fi's need to be properly nested in the replacement texts of macros. (Having these two types of nesting independent of each other has proved to be important in many existing macro processors. Caveat implementor.)

People who try things like \expandafter\else\if... should be shot. (Unless it turns out that this is useful?)

(119) Another token-list parameter \tokens is definable like \everypar. Its only use is for things like \the\tokens (which, in an \xdef, emits the current value of \tokens without further macro expansion).

(120) Another test, \ifcat, is like \if but it tests the catcodes of the characters, not their ascii codes.

(121) \everymath{...} inserts its tokens into TEX's scanner just when non-display math mode has been entered. \everydisplay{...} does likewise, but for displays. For example, \everymath{\fam0} sets up family 0 instead of family −1 as the default for letters. [The default used to be +1; starting with version 0.96 it's −1.] You can also use \everymath to redefine active characters that you want to behave differently in math mode.

(122) \futurelet\a followed by tokens b and c has the effect of "\let\a= c" followed by tokens b and c. You can use this to look ahead at the next token after a macro; it's for hackers. (I put it in because it is easy and because it might allow me to solve some problem next year.)

(123) \prevgraf is an internal state variable something like \prevdepth and \spacefactor. It represents the number of lines in the most recently completed paragraph or partial paragraph. You can use it to provide "memory" between paragraphs with respect to \parshape and \hangindent. For example, if you say "\indent \prevgraf=15 This new paragraph..." the new paragraph will be formatted as if 15 lines had already been completed. You can look at \the\prevgraf to see how big the last paragraph was.

(124) \endinput (in any mode) forces end-of-file whenever the next line of \input has been fully read. This omits the all-blank line that is normally appended at the end of a file that terminates normally. \input is now allowed in any mode, not just vertical mode.

(125) \hangindent (dimen) no longer looks for keywords "for" and "after"; it's just like all of TEX's dimen parameters. \hangafter is another parameter that provides the missing information. Suppose \hangafter equals n. If n is nonnegative, hanging indentation applies to lines $n+1$, $n+2$, ...; otherwise hanging indentation applies to lines 1, ..., $|n|$. For example, the old "\hangindent 10pt for 2" now becomes "\hangindent=10pt \hangafter=-2". The old "\hangindent 20pt" (without "for" or "after") is unchanged, because TEX sets \hangafter=1 after each paragraph and whenever entering internal vertical mode. [Reason for this change: It became all too obvious that the old system was a kludge, when I tried to describe it in the new manual.]

(126) \clubpenalty (a new parameter) applies after the first line of a paragraph; \widowpenalty before the last line. (These used to both be called \widowpenalty. Geoffrey Glaister, U. of Calif. Press 1979, *Glossary of the Book*, distinguishes in this way between "club lines" and "widow lines". The

TEX manual sets \clubpenalty=10000 to suppress breaks after the first line of paragraphs that have dangerous bend on their first two lines.)

(127) \jobname returns the name of this TEX job. For example, if the output goes to file foo.dvi, \jobname is "foo".

(128) \pagetotal and \pagegoal are internal dimensions representing the natural height and goal height of the current page. Previously you could see these only as a result of \showlists; now you can even set them. If a page is empty, \the\pagetotal is 0pt and \the\pagegoal is -1000pt. Setting \pagegoal has no effect on an empty page, since \pagegoal gets set to \vsize as soon as the page gains its first item. Besides \pagetotal, there is also \pagestretch, \pagefilstretch, \pagefillstretch, \pagefilllstretch, and \pageshrink, denoting the total flexibility of glue on the current page; and \pagedepth is the depth.

(129) \deadcycles is an internal integer that is set to zero on each \shipout and increased by 1 each time your \output routine is invoked. At the \end, null boxes will be ejected until the current page and recent contributions are empty and \deadcycles=0. Thus, you should set \deadcycles=0 yourself if you have an unusual output routine.

(130) \insertpenalties is the sum of penalties from split insertions and floating insertions. \floatingpenalty is the amount added to \insertpenalties when an insertion is being held over following one that is split. \insertpenalties is the number of heldover insertions during the time an \output routine is active. (All of this should become clear in the new user manual ...)

(131) New integer parameter \globaldefs (is normally zero). If positive, all definitions are global; if negative, all definitions are local even when they are prefixed by \global. (Of course, some definitions are inherently global and will stay that way: \texinfo is global, and so are things like \prevdepth, \spacefactor, \pagegoal, \deadcycles.)
Example of use: \def\GLOBAL#1{\begingroup\globaldefs=1 #1\endgroup} allows you to say \GLOBAL\tracingall.

(132) \mathchoice{D text}{T text}{S text}{SS text}, where the texts are arbitrary math mode formulas, is now allowed in math mode. TEX will use whatever text fits the style that eventually governs this part of the formula.

(133) \limitswitch is replaced by three primitives \displaylimits, \nolimits, \limits. The last of these takes the superscript and subscript and puts them as limits above and below the operator; the first (which is the default) does this in display style only; \nolimits gives ordinary subscripts and superscripts. (Now for the first time you can display limits in superscripts, using \scriptfont3, if you really want to.) To change old files, you probably need only to replace \limitswitch by \nolimits; also redefine \int and \oint to include \nolimits.

(134) \chardef is analogous to \mathchardef: instead of saying \def\foo{\char⟨number⟩} you can say \chardef\foo. Furthermore, both \chardef and \mathchardef define control sequences that can be used as integers. For example, the allocation command \newcount\exno in plain TEX used to produce \def\exno{28 } (for example), but now it produces \chardef\exno=28. This is more efficient in both time and space. A \chardef is limited to the range 0..255; \mathchardef has the range 0..32767.

(135) \input is treated in the syntax level, so you can say \expandafter\foo\input bar (in case you want to input a file that doesn't start with any particular macro). But recursive use (e.g. \input\input f to get one file name from another) isn't allowed.

(136) \unbox becomes \unhbox and \unvbox (which are capable of unboxing only hboxes and vboxes, respectively). Previously you could say \unbox in either horizontal or vertical mode, but you had to be in the right mode to match the box; now if you say \unhbox in vertical mode, or \unvbox in horizontal mode, TEX switches to the correct mode before doing the unboxing. (The reason the new syntax is needed is that TEX couldn't otherwise provide this feature without parsing the number that follows the \unbox, but then when switching to horizontal mode would have to insert \everypar before the stuff that was already parsed; it gets pointlessly complicated that way.)

(137) \unhcopy ⟨number⟩ and \unvcopy ⟨number⟩ are added too; they produce a copy of the inside of a box. One application is that \struts are now more efficient: You can define a strut box that contains a

\vrule, and "\def\strut{\unhcopy\strutbox}". Previously, TEX had to scan the keywords "height" "width" and "depth" and their dimensions whenever it appended a rule; but rules of width zero make good struts since they take less memory than boxes. Furthermore, with this new system you can use a strut to begin a paragraph (even if a box comes first in the paragraph), instead of saying "\ \unskip" or "\hskip0pt".

(The following changes have been introduced in version 0.96.)

(138) \delimitershortfall is new name for \delimiterlimit.

(139) Blank space is no longer ignored after a control sequence name of the form ⟨escape character⟩⟨nonletter⟩. For example, this applies to \/ and \\ and \" and \´ and \`. However, any blank spaces after ⟨escape character⟩⟨space⟩ are ignored, as part of the general rule that consecutive blanks are treated as single blanks.

(140) \crcr acts like \cr except if it appears at the very beginning of an alignment line; in the latter case it is ignored. Thus, if you're writing a macro that generates alignment, you can put in a \crcr, which will cover for users who forgot the final \cr in their argument to the macro.

(141) Trailing blanks are removed from all input lines. This makes TEX fully compatible between IBM-like installations (where lines have fixed width) and DEC-like ones (where lines are terminated by carriage returns).

(142) The glue above and below displayed equations no longer needs to be the same. (And it shouldn't be, because a change to \baselineskip inside the $$'s will affect the top distance but not the bottom one ....) \abovedisplayskip and \belowdisplayskip take the place of \dispskip. \abovedisplayshortskip and \belowdisplayshortskip take the place of \dispaskip and \dispbskip.

(143) \postdisplaypenalty is the penalty for page breaking after a display (just before the \belowdisplayskip glue). \predisplaypenalty is the penalty for breaking before a display (just before the \abovedisplayskip glue). \displaywidowpenalty is the penalty for breaking between lines of a paragraph in such a way that there's just one line before a display.

(144) \tracingrestores=1 will print a symbolic indication of what values have been restored at the end of a group, on your log file, if TEX has been compiled with its "stats" switch on. For example, consider '\lineskip=3pt{\lineskip=4pt.....}'; when the } is sensed, your log file will say '{restoring \lineskip=3.0pt}'.

(145) \iftrue and \iffalse are new conditional tests that always go one way or the other. To use them, you can say, e.g.,

```
\let\iftitlepage=\iftrue
```

and later on make tests like

```
\iftitlepage ... \else ... \fi.
```

Plain TEX provides a macro \newswitch such that, e.g., \newswitch{titlepage} defines two new control sequences \titlepagetrue and \titlepagefalse; the first of these expands to \let\iftitlepage=\iftrue. You should use this convention instead of saying "\let\iftitlepage=\iftrue" explicitly in your manuscript, because the "\iftrue" in the latter will get TEX mixed up if it is skipping over unexecuted conditional text.

(146) \everyvbox{...} specifies a list of tokens to be inserted just after the opening left brace of \vbox, \vtop, and \vcenter. Similarly, \everyhbox{...} specifies a token list for the beginning of \hbox. One use of these might be to "shut off" macros that were "turned on" by \everymath and \everydisplay.

(147) \romannumeral ⟨number⟩ joins \string⟨cs⟩ and \number⟨number⟩ and \jobname as ways to emit text. For example, \romannumeral 324 comes through TEX's scanner as "cccxxiv" (unless macros are not being expanded); count5=-123, \number\count5 comes through as "−123". If the number is zero or negative, \romannumeral emits no text.

Some new macros of PLAIN.TEX (148–152).

(148) The following macros are superior to their "basic.tex" counterparts, and they are going to be explained in the new manual. Meanwhile, until you see the new manual, the only way to figure out how to best use them is to look at the way they are defined in plain.tex ....

(149) Instead of \ldotss use just \ldots, if followed by something other than a closing delimiter inside a formula; instead of $\ldotss$, use \dots; instead of $⟨something⟩\ldotss⟨something⟩$ in other cases, use "\ldots\,".

(150) Instead of \twoline, use \displaylines.

(151) Instead of \chop, use \smash.

(152) Instead of \cpile, use \matrix.

(The following changes have been introduced in version 0.97.)

(153) Undelimited macro parameters will not be set to "space" unless you explicitly say "{ }". For example, in \def\a#1#2#3.#4{...}, parameters #1, #2, and #4 are undelimited, while #3 is delimited by a period. In TEX80, if you said "\a x y z. w" the result was #1=x, #2=space, #3=y z, and #4=space; this caused confusion. Similarly if you said \a{...} at the end of a line, you got a space for #2 because of the carriage return. In TEX82 versions > 0.96, you will get #1=x, #2=y, #3= z (including the space), #4=w.

(154) \everyjob{⟨tokens⟩} specifies something that will be put into TEX's scanner at the beginning of every job. (This is useful only in a format that is \dumped, because otherwise the job has already started!)

(154) \ifeven disappears; \ifodd⟨number⟩ takes its place, where ⟨number⟩ is the number being tested (not the number of a count register). Thus, where you used to say

\ifeven *n* alpha \else beta \fi

you now say

\ifodd\count *n* beta \else alpha \fi

(156) \if and \ifcat no longer give error messages in cases like "\if\par\let"; unexpandable control sequences are regarded as having character code 256 and category code 16. Thus, you can use \if without fear of an error stop when the user's input is unexpected.

(157) \leaders will align from the left of the smallest enclosing box, rather than from the largest. Thus, for example, if you \moveright a box that contains leaders, the leaders will move right with it.

(158) If you say && in the preamble to \halign and \valign, the preamble will infinitely repeat the material to its right. For example,

\halign{\indent#\hfil&&\hfil#\hfil&#&\cr is like
\halign{\indent#\hfil&\hfil#\hfil&#&\hfil#\hfil&#&\hfil#\hfil...\cr

for as many columns as are actually used. This makes it possible to write, e.g., a \matrix macro that has no maximum number of columns. (If you say && twice in the same preamble, the second && is erroneous.)

(159) \read will read several lines, if necessary, until the number of left braces equals the number of right braces. "\outer" control sequences should not appear in the input.

(160) \uppercase and \lowercase now apply to all character tokens in the token list, including active characters. (Previously they applied only to characters of categories 11 and 12.)

(Updates to the April 1983 manual)
(These changes were installed in version 0.98.)

(161) \read *n* to \cs will not print an explicit prompt message if $n < 0$. \write *n* {...} will print only on the log file (not the terminal) if $n < 0$.

(162) New parameters \hoffset and \voffset will offset the output position. For example, \hoffset=.5in shifts subsequent output right by half an inch; \voffset=1.5in shifts it down 1.5in with respect to its normal position. (The \shipout command adds these offsets to all coordinates of things that it is shipping out.)

(163) Here is a fairly major change with respect to TEX macro programs. The notation for parameters and registers is unified and made more efficient.

1) You don't need \the anymore when using a parameter in an expression. Thus, \hbox to\hsize works. (So does \hbox to\the\hsize, since a redundant \the is allowed.) Same for \the\texinfo, \the\lastskip, etc.; \the is mainly to be used inside \xdef and such things, now. \minusthe disappears.

2) \setcount is eliminated, you just write \count. Similarly \setdimen and \setskip and \setmuskip are eliminated.

3) \advcount, \advdimen, etc. are now \advance\count, \advance\dimen, etc.; similarly there is \multiply and \divide. These will work with parameters as well as registers; e.g., "\advance\abovedisplayskip by 3pt".

4) To compensate for the extra token (\advcount split into two), there are new commands \countdef, \dimendef, \skipdef, and \muskipdef (analogous to \chardef and \mathchardef). For example, \countdef\c=5 makes \c a shorthand for "\count5".

5) "dm" is eliminated as a unit, but instead you can write .5\hsize or .5\dimen⟨number⟩. "vu" and \varunit are also eliminated, as they are now unnecessary.

The \newcount...\newmuskip macros now give their results in terms of \countdef...\muskipdef, not \chardef. One consequence is that \dimendef makes a control sequence behave syntactically like a new dimen parameter. For example, it would now be possible to say "\newdimen\hoffset" and to incorporate \hoffset in one's \output routine, thereby making the new \hoffset parameter unnecessary. Previously, you would have had to tell users to give values to symbolic dimensions by constructions like "\def\hoffset{.5in}" because of the incompatibility between register syntax and parameter syntax. (I apologize for letting this shameful incompatibility creep in, years ago. Soon it will be gone forever.)

(164) Here's a simple way to update old macros so that they work with version 0.98:

Case (1), you have not used \newcount, \newdimen, etc.:
Replace \setcount by \count, \setdimen by \dimen, \setskip by \skip, and \setmuskip by \muskip.
Replace \advcount by \advance\count, \advdimen by \advance\dimen, \advskip by \advance\skip, \advmuskip by \advance\muskip.
Replace \multcount by \multiply\count, \multdimen by \multiply\dimen, \multskip by \multiply\skip, \multmuskip by \multiply\muskip.
Replace \divcount by \divide\count, \divdimen by \divide\dimen, \divskip by \divide\skip, \divmuskip by \divide\muskip.
[That doesn't give the most efficient programs, but it will tide you over until you have time to rewrite things.]

Case (2), you have used, e.g., \newcount\foo:
Replace \setcount\foo by \foo, \the\count\foo by \the\foo, \advcount\foo by \advance\foo, \multcount\foo by \multiply\foo, \divcount\foo by \divide\foo. Similarly for \newdimen et al.

(165) \everycr{...} inserts its tokens just after TEX has processed \cr or \crcr. (Except if the \crcr was ignored because it came right after \cr.)

(166) \hyphenchar⟨font⟩=⟨number⟩ defines the character to be used for hyphens in that font. If the number is negative or greater than 255, no hyphenation will be done. Default is *'55*.

(167) \skewchar⟨font⟩=⟨number⟩ defines a character to be used to position accents in math mode. (This makes it unnecessary for a user to refer to a long, horrible "\skew" table like that in the April draft of the manual!) When TEX puts a math accent over a character, it shifts the accent to the right by the amount of kern between that character and the skew character. Default is −1.

(168) Furthermore, when a single character is accented, the subscripts and superscripts of the accented combination are now attached exactly as they would have been without the accent. (Previously, "\hat A↑2" put the 2 too high, and "\hat P_2" put the 2 too far away from the P; now both cases have been fixed.)

(169) New syntax: \texinfo⟨font⟩⟨number⟩ becomes \fontdimen⟨number⟩⟨font⟩. This means that you can define, e.g., \fontspace to be an abbreviation for "\fontdimen2" and then you can set \fontspace⟨font⟩=10pt (analogous to \skewchar).

(170) \meaning and \noexpand are new primitives that affect macro expansion:

\meaning⟨token⟩ expands to the sequence of characters that would be displayed on the terminal by the existing commands

\let \test = ⟨token⟩ \show\test

For example, "\meaning A" expands to "the letter A" (a sequence of twelve character tokens, all type otherchar except the spaces). After \def\A#1B{\C}, "\meaning\A" will expand to "macro:#1B->\C " (a sequence of fourteen tokens). You can use this in an emergency when \if and \ifcat and \ifx don't tell you what you need to know about a token.

\noexpand⟨token⟩ produces the token, but changes its meaning to the meaning of TEX's \relax primitive if that token would ordinarily be expanded. Thus, it's now easier to suppress macro expansion; and you can use \noexpand fruitfully after \if or \ifcat when you are testing the nature of an unknown token (e.g., a token that has been found by \futurelet).
Note the following: \catcode`\~=13 \ifcat\noexpand~\noexpand~true\fi yields "true", but \ifcat\relax\noexpand~true\fi doesn't; i.e., the *TEXbook*'s conceptual model of token lists, in which an active character carries category code 13 as a "subscript", is now implemented.

(171) \afterassignment⟨token⟩ saves the token and reinserts it into TEX's input mechanism after the next assignment has been performed. An assignment is a \def or \let or a command that assigns a value to a control sequence or internal register; a complete list of assignments will appear in Chapter 24 of *The TEXbook*. The main intended use of \afterassignment is this: If the replacement text of a macro ends with "\afterassignment\continue \dimen0=", then TEX will assign to \dimen0 whatever ⟨dimen⟩ follows the macro, after which it will perform \continue. Similarly, you can end with an assignment that causes TEX to parse ⟨glue⟩ or a font name or even a \parshape.

(Changes to PLAIN.TEX conventions since the April manual)

(172) The \openup macro now takes a dimension argument without braces. It is recommended to give the amount in terms of a \jot, which plain TEX sets to 3pt. Thus, where the manual says "\openup{3pt}", you now say "\openup 1\jot"; where it says "\openup{-3pt}", you now say "\openup -\jot"; and fractional amounts like "\openup .5\jot" are legit.

(173) The character for "ties" between words has been changed from @ to ~. The next generation of fonts will have an at sign in the normal ascii place, so @ will not be mentioned specifically as a special character in the *TEXbook*. But for the time being (until we have the new fonts), you still need to say \@ to get an at sign from the math symbols font. Some day there won't be an at sign in that font, but you'll have bold and italic at signs in your bold and italic fonts!

Incidentally, although this doesn't affect users, PLAIN and LATEX and AMS-TEX will be using @ as a letter in the names of control sequences that aren't supposed to be easily redefinable.

(174) In math formulas, " is no longer used for double prime; the notation is f''(x) instead of f"(x), and f'''(x) will also work. In other words, " is now unconstrained, while a sequence of n apostrophes is converted into ↑{\prime...\prime} (there are n occurrences of \prime).

(Changes installed in version 0.99.)

Version 1.0 is almost here! Just a few more touchups ...

(175) "wd", "ht", and "dp" are changed to \wd, \ht, and \dp. (This is something like the recent change where "dm" was changed to \dimen.) That means you don't have to say "1wd0" to get the width of \box0, you just say "\wd0". Furthermore, you can now assign new dimensions to a box, instead of relying on

trickery: For example, \ht0=2pt changes the height of \box0 to 2pt, independent of what the height was formerly. (But it has no effect if box0 is void. The prefix \global is ignored if you happen to say \global\ht0=2pt.)

(176) \tokens is being generalized to a set of 256 registers called \toks0 through \toks255. And there is "\toksdef". Plain TEX will contain a \newtoks allocation macro, of course. (Thus you get symbolic names for lists of tokens that are unlike macros since you can insert them into edefs and messages with one-level expansion only.) [Note: Say "\toksdef\tokens=0" to make your previous programs work.]

(177) \insert, \vadjust, and \mark are allowed in restricted horizontal mode and in math mode. However, they won't always migrate outside their boxes into the main vertical list; migration from an hbox happens only if the hbox was typeset on the outer level of a vertical list (most uses of \centerline fit this description) or if the hbox was formed for an \halign entry. Migration from math mode happens only if the item is on the outer level of a math formula (not in a subformula) and if the formula is displayed or if it appears in a paragraph or hbox that allows migration. Otherwise you have to unbox them if you really want to use them. This makes \hbox and \vbox work approximately the same.

(178) \font is now acceptable as a font identifier in contexts like "\fontdimen6\font" and "\hyphenchar\font" and "\textfont0=\font". It denotes the current font.

(Version 0.999.)

(179) \leaders in a horizontal list now have height and depth; in a vertical list they have width.

(180) The word "by" is optional in the \advance, \multiply, \divide commands.

(181) \font\foo=name scaled n says to load the font at $n/1000$ times its design size. Plain TEX has a macro \magstep(n) such that \magstep0=1000, \magstep1=1200, \magstep2=1440, \magstep3=1728, \magstep4=2074, \magstep5=2488, others undefined. Also \magstephalf=1095. We will make and distribute fonts that are magnified in such steps.

The \magnify macro has been renamed \magnification and it is designed to to be used without braces:

\magnification 1200 or \magnification=1200 or \magnification \magstep1

are all equivalent. Note that if you use fonts only at \magstep0 or \magstephalf, (e.g., \font\ff=cmr7 scaled \magstephalf) you can say \magnification\magstephalf and you will still stay in the existing font family.

\magstep 0 leaves 10pt as 10pt;
\magstephalf makes 10pt into 11pt, (very nearly);
\magstep 1 makes 10pt into 12pt;
\magstep 2 makes 10pt into 14pt, (a teeny bit more, actually);
\magstep 3 makes 10pt into 18pt, (a teeny bit less, actually);
\magstep 4 makes 10pt into 21pt, (a teeny bit less, actually);
\magstep 5 makes 10pt into 24pt, (a teeny bit more, actually).

We will make the sixteen fonts of plain TEX available in seven magnifications (including \magstep 0, of course!). Other fonts will be made in four magnifications. Any font can, of course, be made in any magnification, subject to economic considerations.

(182) New parameters to free TEX from its character set a bit more:

\escapechar is the character inserted before control sequence names by \write and \string, and when TEX displays token lists

\defaulthyphenchar and \defaultskewchar, the values assigned to \hyphenchar and \skewchar when a font is loaded

\endlinechar, the character placed at the end of an input line.

In the case of \escapechar and \endlinechar, no character is used if the value is negative or greater than 127. Normally \escapechar=`\\, \defaulthyphenchar=`-, \defaultskewchar=-1, and \endlinechar=`\^^M ((return)).

(183) A new tokenlist parameter \errhelp is added. TEX will do the equivalent of \immediate\write16{\the\errhelp} if the user types "h" in response to an \errmessage stop. Note: To conserve TEX's memory space, you should do something like the following [plain TEX will provide a macro]:

```
\edef\next{\csname This is my help message.\endcsname}
\errhelp=\expandafter{\next}
```

Then the \errhelp token list is only one token long, and when you write it TEX will print the long control sequence name like this:

```
\This is my help message.
```

The control sequence name goes into TEX's compact string memory, where TEX's own help messages are stored. By contrast, if you had said

```
\errhelp={This is my help message.}
```

you would have taken up space at the rate of one token per letter!

(184) When TEX writes a token list to a file and the character \newlinechar occurs, TEX will start a new line instead of writing that character. (Thus, for example, you can give two-line help messages.) Plain TEX will set \newlinechar=-1, ignoring this feature.

(185) Assignments like \toks2=\toks4 and \everypar=\everymath are now allowed.

(186) When \edef and \message etc. are expanded, expansion is no longer inhibited after \def; \noexpand should be used to suppress expansion there.

(187) \the is now treated as an expandable command; it still expands only one level in \edef and \mark and \write, etc. If you now say, e.g., "\hbox to\the\hsize" you lose lots of time w.r.t. "\hbox to \hsize", because the former expands \the\hsize to a sequence of tokens and parses them, while the latter has instant access to \hsize. Conversely, \the is more efficient than \number in \edef, because it avoids trying to expand the digits after they are generated. Previous features \the\the and \the\tenrm are no longer allowed; there's a new command \fontname⟨font⟩ which expands to the name of the specified font. (The rules for macro expansion are now somewhat cleaner and more consistent, and the formal syntax is now nicer.)

(188) \unhbox and \unhcopy are now allowed in math mode provided that the box in question is void. This means that you can say \unhcopy\emptybox in any mode, and it puts you into horizontal mode if you were in vertical mode. (This is more efficient than \ \unskip.)

(189) \aftergroup⟨token⟩ puts the token into TEX's input after the current group ends. If several \aftergroup commands occur in the same group, they will be performed in order; e.g., {\aftergroup\a \aftergroup\b} yields \a\b.

[This is the historic and climactic final extension to TEX.]

(Changes after July version of TEXbook.)

(190) There's no longer a blank line inserted at the end of an \input file. The paragraph at the bottom of page 47 of the manual has been changed to:
[danger][danger] If TEX has nothing more to read on the current line, it goes to the next line and enters state N. However, if \endinput has been specified for a file being \input, or if an \input file has ended, TEX returns to whatever it was reading when the \input command was originally given. (Further details of \input and \endinput are discussed in Chapter 20.)

(191) \ifhbox and \ifvbox join \ifvoid (see page 210). Thus, you can test for three possible states of a box register, which always is either void or contains an hbox or a vbox.

(192) \lastkern and \lastpenalty are analogous to \lastskip;
\unkern and \unpenalty are analogous to \unskip.

Index to Differences Between TEX82 and TEX80

References are to item numbers in the TEX82/TEX80 differences list.

METAFONT Errata

As of 09 September 1983

This document lists changes to the **METAFONT** manual, occurring since its publication as a Stanford report in 1979 (which is the same as the version published by Digital Press in December 1979). The programs and font specifications in Appendix E have been thoroughly revised, and the Stanford report CS780 (*The Computer Modern Family of Typefaces*) shows a more up-to-date version; however, some significant changes to that appendix are listed below, and the real truth appears in the computer files described in `FILES.INF[FNT,DEK]` on the Stanford SAIL computer.

Page 4, line 8, TEX should have been told not to break at that hyphen!

Page 24, line 6, change "h_0 units high," to "l_0 units high, where l_0 is the current value of **lpenht**,"; line 12, change it to "**lpenht** 25; **rpenht** 25;"

Page 28, table, change h_0 to l_0 in the **lpen** column and to r_0 in the **rpen** column.

Page 34, line –22, change "`tfxmode`" to "`tfmmode`".

Page 34, lines –17 and –20, and page 35, lines –1 and –6, change "`points`" to "`pts`" (total of five changes).

Page 38, line 17, change "`DRAGON.TFX`" to "`DRAGON.TFM`".

Page 48, table, change h_0 to l_0, r_0 in the **lpen,rpen** column.

Page 53, line 8, change $(x+1, y)$ to $(x, y+1)$.

Page 61, insert a new "dangerous bend" just before Chapter 9:

> **METAFONT** has a limited memory available for storing subroutines. If you know that some subroutine, say "*s*", is no longer going to be needed, you can remove it from **METAFONT**'s memory by writing "**subroutine** *s*:.".

Page 61, fifth line of Chapter 9, change "points" to "point" and "signs" to "sign".

Page 65, after line 14, insert:
charwx and **charwy** are used in an analogous way to specify the x and y coordinates of the two-dimensional width vector of a character that might be rotated. These parameters are used when preparing fonts and font information files for the Press world as described in Appendix P.

Page 65, before line –11, insert new real parameters:

> **designsize** specifies the nominal body height of the font in units of printers' points: a "10 point font" is a font whose **designsize** is 10. This parameter is used in constructing both the TEX and Press font information files.
>
> **hresolution** and **vresolution** specify the resolutions of the raster in the x and y directions respectively. This information is stored in the font file intended for use by a Dover Press printer (**ocmode**). These resolutions should be given in pixels per point.
>
> **magnification** (normally 1.0) is a pure number that tells **METAFONT** the amount by which the current font is being photographically expanded. If the **magnification** is not unity, it affects the interpretation of the **designsize** and **resolution** parameters as explained in Appendix M.
>
> **rotation** (normally 0) specifies the *rotation* attribute for use by the Press world, in units of degrees. (See Appendix P.)

Page 65, lines –9 and –8, change to:

> **hpenht**, **vpenwd**, **lpenht**, and **rpenht** (normally 1) are used to specify the height of each **hpen**, the width of each **vpen**, the height of each **lpen**, and the width of each **rpen**. It is best ...

Page 66, after line 3, insert new integer parameter:

> **fontfacebyte** specifies the encoded value of the *face* attribute for use by the Press world (Appendix P). This value must be an integer between 0 and 255 inclusive. It is stored in TEX's font information file as well as in the Press-oriented output files. The default value is 0.

Page 66, lines 10 and 11, delete "; this information ... XGP".

Page 66, lines 14–19, change to:

crsbreak specifies a y coordinate at which a tall character will be broken into pieces when preparing it for an Alphatype CRS font. If you specify several different **crsbreaks** for the same character, it will be broken into several different pieces. At most 1020 nonzero rows of the raster should appear between breaks. Furthermore, a **crsbreak** of 1000000 is used to indicate a character that is combined with others for special typesetting effects (e.g., parts of a **varchar**); this causes the Alphatype software to take special care to position the character perfectly.

Page 66, before line –8, insert new title parameters:

- ⟨title parameter name⟩ "⟨any desired title⟩"

Some parameters have titles as values instead of numbers. Such parameters, like titles themselves, may not appear in subroutines. Lower case letters in title parameters are converted to upper case letters so that other programs that must deal with such parameters do not need to make case distinctions. The following title parameters are understood by **METAFONT**:

codingscheme describes the correspondence between character codes and character shapes used by the current font. For example, "**codingscheme** "ASCII"" describes an ascii character set. This descriptive string is put into the font information file for TEX. At the moment it is used only in some auxiliary software that reads and writes .TFM files; but since it provides important documentation, wise users will specify the coding scheme explicitly by means of this title parameter.

fontidentifier specifies the *family name* attribute for use by the Press world (see Appendix P). This family name is stored in TEX's font information file as well as in the Press-oriented output files.

Page 66, line –4, change "and **penreset**" to "**penreset**, and **points**".

Page 67, lines 8–10, change "from ... to ... " to "between ... and ..., inclusive";
line –12 (re **penreset**), change "begins." to "begins or a section ends."; before line –11, insert:

points causes **METAFONT** to display labeled points in proof mode.

arrow causes **METAFONT** to mark the reference point of proofmode output so that the figure can be merged with other documents (allowed only when proofmode output goes to Press files).

color causes **METAFONT** to prompt the user for what colors to use in proofmode output (allowed only when such output goes to Press files).

Page 67, line –9, change "These" to "If **points** is turned on, these";
lines –6 thru –4, delete "Thus you ... section."

Page 68, after line 2, insert new output mode:

ocmode causes **METAFONT** to output a file of font images in the format required for Dovers and some other Press printers.

Page 68, line 11, change "**tfxmode**" to **tfmmode**";
after line 12, insert new output mode:

dotwdmode causes **METAFONT** to output a file of width information that programs in the Press world need when they use a font. (This mode can't be called "**wdmode**" because that word starts with a "w".)

Page 68, line 17, change "**tfxmode**" to "**tfmmode**".

Page 69, line 11, change ".XGP, .CHR, .TFX" to ".XGP or .DVI or .PRESS, .CHR, .TFM, .OC, .WD"

Page 69, line 12, change "**tfxmode**" to "**tfmmode, ocmode, dotwdmode**".

Page 71, line 22, change "**range**." to "**range** (⟨coordinates⟩)."

Page 73, replace last four lines by:
`! Ligature/kern program didn't end.`
The final entry of your final (lig instruction list) ended with a comma. Proceed; the comma will be ignored.

Page 74, a new error message:
`! lpen height too small, set to 1.`
You shouldn't try to make **lpenht** less than 0.5. Proceed, and it will be set to 1.

[A similar message about **rpenht** goes on page 77, and the wording of the similar messages for **hpenht** and **vpenwd** should be shortened to match.]

Page 74, new types of overflow:

`brksize`	number of **crsbreak** points in a character;
`initblocks`	number of preamble blocks in binary font file

Page 77, new error messages:
`! Square root of -`⟨constant⟩`, replaced by 0.`
You tried to take the square root of this negative number. Proceed, and its value will be zero.
`! Title expected, command flushed.`
A title parameter to **METAFONT** should begin with a double-quote mark. Proceed, and all tokens up to the next semicolon or period will be ignored.

Page 79, new error messages:
`! Varchar can't be in the middle of a charlist.`
A character code specified as a **varchar** is allowed in a **charlist** only at the end. Proceed; if the message says "`command flushed`", the remainder of this **charlist** is ignored, otherwise the **varchar** specification for the current character code is ignored.
`! Varchar can't have ligature/kern.`
A character code specified as a **varchar** must not appear as a label in a ⟨lig instruction list⟩. Proceed; if the error message says "`command flushed`", the remainder of this ⟨lig instruction list⟩ will be ignored, otherwise the **varchar** specification for the current character code is ignored.

Page 82, after line –3, insert:

fontidentifier "CMR"; *ptsize* = 10;

Page 83, line 14, change "eight" to "nine".

Page 83, after line 15, insert new "*p*" variable:

ptsize is the nominal font body height, 10 points in our example.

Page 84, line 5, interchange "height" and "width".

Page 85, line 14, change $\frac{37}{23}$ to $\frac{23}{37}$.

Page 86, line –4, delete the "and" after the semicolon;
line –2, change "drawn." to the following:
drawn; and mode 3 generates a font for a Dover Press printer with a resolution of 384 pixels per inch, displaying nothing.

Page 87, bottom line, "$c \leftrightarrow px$" should be "$m \leftrightarrow px$".

Page 88, lines 2–3, change "Fortunately ... x-height." to:
Variable m is used to stand for the x-height, since a line at this height is traditionally called the "mean line".

Page 88, line 6, change 'a "c" is in row c' to 'an "m" is in row m' (and so on, changing "c" to "m" in lines 7 and 8).

Page 88, line 7, change "pixel ... appears" to "pixels ... appear".

Page 88, lines –9 through –5:

w_8, the curve height;
w_9, the upper-case stem height.

Note that the last four of these variables have no "*p*-variable" equivalent; they satisfy the approximate relation

$$\frac{w_6}{w_0} \approx \frac{w_7}{w_1} \approx \frac{w_8}{w_2} \approx \frac{w_9}{w_4} \approx aspect.$$

The **hpenht**, **lpenht**, and **rpenht** are w_6 and ...

Page 90, lines 16–19 of **Fig. E-2** should be indented.

Page 91, lines 12 and 16 of **Fig. E-3**, change "counter" to "bowl".

Page 91, line –11, change 'a "W" too.' to '"M" and "W".'

Page 91, replace the last five lines by:

```
designsize ptsize;
mag = 1.0; magnification mag;
if mode = 0: proofmode; drawdisplay; pixels = 36mag; blacker = 0;
else: if mode = 1; fntmode; tfmmode; chardisplay; pixels = 3.6mag; blacker = 1.2;
      else: if mode = 2: crsmode; tfmmode; titletrace; pixels = 73.7973mag; blacker = 1;
            else: if mode = 3: ocmode; tfmmode; dotwdmode;
                               pixels = 384 · .013837 · mag; blacker = 0.75;
                  else: input mode;                                    %get mode
                  fi;                                                  %information from
            fi;                                                        %another file
      fi;
fi;
hresolution pixels; vresolution pixels;
```

Page 92, line 3, change comment to read:

```
                                                   %the vertical size of the font, normally equal to ptsize
```

Page 92, lines 5 and 17, change "c" to "m".

Page 92, lines 12–15, change to:

```
w6 = round(pixels · pw · aspect + blacker);
w7 = round(pixels · pwi · aspect + blacker);
w8 = round(pixels · pwii · aspect + blacker);
w9 = round(pixels · pwiv · aspect + blacker);
hpenht w6; vpenwd w0; lpenht w6; rpenht w6;
```

Page 92, line 23, change "3*pu*,3*pu*" to "3*pu*,2*pu*"

Page 92, line –2, delete "**charic** *italcorr*", also delete line –1.

Page 93, lines 1-8, change to:

```
tu = pu · pixels; uw = charuw − sc · (lftcorr + rtcorr);
if fixwidth = 0: moduw = uw;
else: moduw = 9; new italcorr; italcorr = 0;
fi;
r = charuw · u = round((moduw · tu − 2) · charuw/uw);
charic italcorr; charwd moduw · pu; chardw moduw · tu;
incx round(−sc · lftcorr · u);
if mode = 0: call box(round sc · lftcorr · u);
```

Page 93, line 10, "guidelines";
line 19 should be

```
y5 = y6 = m; draw 5. .6;                                   %mean line (x-height)
```

Page 96, change "*ru*" to "*pu*" in 8 places.

Page 96, line –8, change ' 'ff' to ' "ff';
line –13, change "example" to "particular example";
line –15, change "**tfxmode**" to "**tfmmode**".

Page 99, lines 1–4, change the sentence to read as follows:
If this is not zero, it denotes the amount of space in points whose multiples will be used for all spacing in math formulas: the otherwise-specified conventions for thin space, thick space, `\mskip`, etc., will be changed so that there is no stretching or shrinking, and the amount of space will be increased in magnitude if necessary to make it a multiple of the math space.

Page 100, line –11, delete "in lieu of a **charic** command";
lines –8 through –6, change the sentences to read as follows:
For example, the extensible left parenthesis symbol in font **cmathx** has been defined by "**varchar** ´060, 0, ´100, ´102"; the second expression is zero because a parenthesis doesn't have a middle component. (The code *c* itself may appear as one of the four components of the **varchar** command for *c*, but this is not required.)

Page 100, line –1, delete ", 0" at the end of the display.

Page 101, lines 3–11, change to read as follows:
... ´060 (the end of the list). Suppose that TEX does reach ´060. As we noted above, the code ´060 in

cmathx represents a built-up character. Thus, TEX can use the pieces specified in that **varchar** command to construct a left parenthesis of the appropriate size. If the last character in a charlist isn't a **varchar**, then TEX will use it whether or not it is large enough. For example, the slash symbols in **cmathx** are specified by "**charlist** ´016, ´036, ´054" where character ´054 is not a **varchar**, but is simply the largest slash present. A **charlist** in general consists of (expression)s (usually constants) that should round to character codes between 0 and 127. The last character in a **charlist** may be a **varchar**, but all of the other characters in the list must be non-**varchars**. In addition, none of the characters in a **charlist** should have a ligature/kern program, since TEX ...

(**NOTE INCOMPATIBLE CHANGE** reflected in the last paragraph: **charlists** don't end with 0 any more unless character 0 is the last in the list.)

Between pages 101 and 102, insert two new Appendices (see below).

Page 102l, new entry "Baseline, 7–8".

Page 102r, first page for **crsbreak** should be 66, not 67; new index entries:
l_0, 24, 28, 48.
lpenht, 24, 65, 74, 88.

Page 103l, add p. 72 under "**hpenht**".

Page 104l, new index entries:
r_0, 28, 48.
rpenht, 24, 65, 77, 88.

Page 104r, new index entry: **subroutine**, 55;
change "**tfxmode**" to "**tfmmode**";
add p. 80 under "**vpenwd**".

****

Appendix M: Producing magnified fonts

The fonts that **METAFONT** produces are intended for use by raster printers. In many cases, the paper or film that comes out of the raster printer is the final product. In other situations, however, the output of the printer is subjected to further processing, perhaps including a photographic scaling. For example, the papers in many conference proceedings are photographically reduced by 25% from the manuscripts that the authors submitted. In such a situation, the output of the raster printer should be thought of as only a middle stage of the printing process. The font images used by the raster printer should be magnified by $33\frac{1}{3}\%$ so that the final result after photographic reduction will be the correct size. The presence of such photographic reduction in the printing process effectively increases the resolution of the raster printing device, at the price of reducing the effective size of the piece of paper that the raster printer can handle.

Some font families have the characteristic that different sizes of the same face are merely obtained by a photographic scaling, or as close to a photographic scaling as the discrete nature of the underlying raster will permit. If we want to submit a paper to a conference and we are using fonts from families of this type, we need only scale up the point sizes that we request. But, as noted in Appendix E, the Computer Modern fonts do not scale photographically: the height, width, stroke widths, and serif dimensions change at different rates to improve the appearance of large and small letters. Thus, if we are using Computer Modern fonts, we really want to print fonts that are photographically magnified, not just switch to fonts that were designed to be larger. **METAFONT** has some built-in mechanisms to assist in the creation of such magnified fonts.

Some of the consumers of **METAFONT**'s output files are smart enough to understand about magnified fonts. For example, both the TEX and Press font information files are written in a scalable format: the metrics in them can be scaled to allow for any photographic magnification of the basic font. This means that only one metric information file is needed per font, even if several different magnifications of that font are in use.

But other consumers of **METAFONT**'s output files don't understand about magnified fonts: most raster printers fall in this class. When producing output files for such ignorant consumers, **METAFONT** itself must scale all distances by the magnification when producing the output file.

The output code in **METAFONT** is able to produce correct output files for both smart and ignorant consumers. But in order to avoid getting **METAFONT** confused, the user of **METAFONT** must ask for magnified fonts in the correct way. To understand the issues involved, think about the structure of a typical **METAFONT** program. We can divide most of the variables in the program into two classes: there are *point* variables, which measure ideal distances, independent of raster resolution; and there are *pixel* variables, which measure distances on the discrete raster. The pixel variables are determined by careful rounding of expressions involving point variables.

In order to construct a magnified font, the **METAFONT** program must be altered so that the values of all of the pixel variables are scaled by the magnification factor (except for the effects of rounding). That has to happen in order to draw magnified characters. But it isn't so clear whether or not the values of the point variables should also be scaled. And it makes a difference whether the point variables are scaled or not, since font metrics such as **designsize** and character metrics such as **charwd** are given in terms of point variables. The **METAFONT** convention is that *point variables should not be scaled when producing a magnified font.* After all, the point variables describe ideal distances; they should be talking about the final result of the entire printing process, not the temporary magnified output of the raster printer.

In order to produce a magnified font with **METAFONT**, it is enough to do the following: (i) scale all pixel variables, except for the effects of rounding; (ii) don't scale any point variables, or any font or character metrics; (iii) tell **METAFONT** what the magnification is by setting the parameter **magnification** to the appropriate real number. Note that some variables have more complex units. You can determine whether or not to scale these variables by combining rules (i) and (ii). For example, a resolution variable has "pixels per point" as its units; since pixels should be scaled and points shouldn't, a resolution should be scaled. In particular, the parameters **hresolution** and **vresolution** should be scaled.

As an example of font magnification in action, the beginning of file `cmbase.mf` in Appendix E shows all that is required. For example, to get a $33\frac{1}{3}$ magnified cmr10 font for a Dover Press printer, one may type `mode=3; mag=1.33333333; input cmr10`. When the file `cmbase.mf` is eventually input, **METAFONT** will stop with the "`Inconsistent equation`" error when it reads the statement "$mag = 1.0$", but you can simply ignore the error and hit (carriage-return). Everything else will work fine.

Another way to incorporate magnification into the program of Appendix E, would be to use, e.g., mode -3 for mode 3 with special magnification. The statement "$mag = 1.0$" could be replaced by

if *mode* < 0: **new** *mmode*; *mmode* $= -$*mode*; **new** *mode*; *mode* $=$ *mmode*;
else: *mag* $= 1$;
fi;

now there will be no "`Inconsistent equation`" error message.

Note that the magnification parameter only has to appear explicitly in two places: in the statement that defines the basic resolution variable named "*pixels*", and in the statement that sets the parameter **magnification**. All of the pixel variables in the Computer Modern programs will be correctly scaled without any other modifications, since they are determined by rounding an expression involving "*pixels*" as a factor.

****

Appendix P: Font information for Press

There is a body of printing software developed within the Xerox Corporation that is centered around a print file format called Press. **METAFONT** can be used to produce fonts for use with Press software and hardware. The Press world has its own conventions about fonts and font information, different from those of the TEX world. These differences and comments on how to resolve them are discussed in the present appendix, which may be regarded as typical of the types of appendices needed to interface **METAFONT** with other conventions that already exist in industry.

The first difficulty arises in the choice of raster coordinates. **METAFONT** believes that the origin point of a character, the $(0,0)$ point of the **METAFONT** raster, is located in the center of a pixel. The Press font world, on the other hand, makes the convention that the origin of a character is located slightly below and left of the center, at the point where four pixels touch, namely at point $(-0.5,-0.5)$ in **METAFONT**'s coordinate system. In other words, if **METAFONT**'s pixel $(0,0)$ is blackened, the lower left corner of this pixel will be at point $(0,0)$ in the Press-oriented editing software. But no change is necessary to the **METAFONT** programs that create the characters.

Press also has a different view about character metrics. In the Press world, characters don't have a height and depth, only a width. But this width is not a single number: instead, the width of a character is a two-dimensional vector. The end of the width vector determines the position on the plane where the origin of the next character should be located, if two characters are adjacent in a string. The use of two-dimensional width vectors allows for fonts that have been rotated by some angle other than a multiple of 90 degrees, as well as for languages that don't go left to right.

METAFONT determines the width vector of a character for Press output modes in one of two ways, as determined by the setting of the "control bit" called **vectorwidths**. The default condition is that **vectorwidths** is not set. In this case, the value specified as the TEX **charwd** is used for the x component of the width vector, and the y component is set to zero. This means that Press will agree with TEX about the widths of characters. If **vectorwidths** is set, however, **METAFONT** will use the values of **charwx** and **charwy** as the x and y components, allowing the user to specify an arbitrary width vector.

The final major difference between the TEX and Press font worlds revolves around the issue of naming a font. In the TEX world, every font has a text name. Each font is stored in a separate file, and the font's text name, supplemented by a standard extension, is used to name that file. Files that contain metric information for a font use the same text name with a different extension. In the Press world, fonts are gathered together into large files called dictionaries. At the beginning of a dictionary is an index that describes what fonts the dictionary contains, together with a description of what kinds of information are present. A font is specified by giving the values of four attributes: the *family name*, *face*, *size*, and *rotation*.

In order for **METAFONT** to produce fonts for the Press world, it must be told the appropriate values of these attributes. **METAFONT** decides upon a *family name* by using the value of the title parameter **fontidentifier**; the *face* is specified by the integer parameter **fontfacebyte**; and the *rotation* is derived from the real parameter **rotation**. The standard file of character width information in the Press world contains metrics that can be scaled to cover any size. For raster fonts, the *size* attribute is determined by taking the distance specified as the **designsize**, and multiplying by the pure number specified as the **magnification**.

****

Experimental features in local METAFONTs

Lyle Ramshaw at PARC is experimenting with splines that control curvature as well as tangents at the key points; watch this space for news about the syntax and semantics of this extension.

Primitive input of binary files is being implemented temporarily with a "**binput**" statement. To use this feature, prepare one or more binary files containing the following: One or more entries of length $2+mn$ consisting of

m	(1 word)
n	(1 word)
(36-bit data)	(mn words),

followed by an entry with $m \leq 0$ to denote end of file. The 36-bit data words specify the bits of a character in m rows, with the first row corresponding to $y = 0$, the next to $y = -1$, etc.; within a row there are n words specifying $36n$ bits from left to right, corresponding to $x = 0$ to $x = 36n - 1$. The settings of **incx** and **incy** will be added to move the raster image; for example, if **incx** $= -18$ and **incy** $= 15$, the first of the mn words specifies the 36 bits for $y = 15$ and $-18 \leq x < 18$. The $36mn$ bits are or'ed in to whatever is already in the raster. (Only **incx** and **incy** are used, not **trxx** etc.) The **METAFONT** statement "**binput**" now means this: If no file for binary input is open, the user will be prompted to supply a file name. Otherwise the next entry from the open file will be used, until the end of file, when the user will be prompted for another file name.

Warning: "**binput**" will not be supported in later versions of **METAFONT**, it will be supplanted by a more compact and machine-independent run-length-encoded format for characters.