PLAIN.TW

Włodek Bzyl **matwb@halina.univ.gda.pl**

Contents

Introduction

However, we will not include a verbatim description, because some parts of that file are too boring, and because the actual macros have been "optimized" with respect to memory space and running time. *D.E. Knuth about* `plain.tex` *format.*

What follows is devoted to the details of the plain TEX format. This file serves two purposes:

(1) As a documentation of the `plain.tex` format. Weaving and texing this document should produce a handy reference.
(2) The division of this web source into 'chunks' should ease creation of other formats tailored to particular applications. Chunks could be easily modified, removed, added, or replaced.

The change file mechanism is not needed in case of TEX language. Change files are used to incorporate system dependent code into source file, but TEX code is already system independent. TEX code could be only 'format dependent' and here change files could be used. Another feature of format file is that it evolves with time, yet some intermediate versions are used for preparation of books, articles etc. All these versions and configurations must be kept well organized, otherwise you are lost. The Revision Control System is the tool that assists with these tasks. With the RCS it is possible, with small overhead, to preserve *all revisions* which evolved from given text document, merge changes made by others, compare different versions, keep log of changes.

This document consists mainly of excerpts from the *TEXBook*, but it is organised around the macros as they appear in the `plain.tex` rather than around the topics as in a user manual. Therefore this document is not a *user manual*, although many definitions are contained here.

The layout of the format

⟨ * ⟩≡

```
⟨ Establish standard category code values ⟩
1  \catcode`@ = 11
⟨ Define commonly used constants ⟩
⟨ Provide programming constructs ⟩
⟨ Allocate registers ⟩
⟨ Assign initial values to parameters ⟩
⟨ Set up text fonts ⟩
⟨ Set up math fonts ⟩
⟨ Provide macros for text formatting ⟩
⟨ Provide macros for math formatting ⟩
⟨ Prepare page for output ⟩
⟨ Read hyphenation patterns ⟩
⟨ Initialize the layout ⟩
2  \catcode`@ = 12
⟨ Identify the format ⟩
```

There are 256 characters that TEX might encounter at each step, in a file or in a line of text typed directly on your terminal. These 256 characters are classified into 16 categories numbered 0 to 15:

Category	*Meaning*	*Default*
0	Escape character	`\`
1	Beginning of group	`{`
2	End of group	`}`
3	Math shift	`$`
4	Alignment tab	`&`
5	End of line	⟨return⟩
6	Parameter	`#`
7	Superscript	`^`
8	Subscript	`_`
9	Ignored character	⟨null⟩
10	Space	␣
11	Letter	`A`, . . . , `Z` and `a`, . . . , `z`
12	Other character	none of the above or below
13	Active character	`~`
14	Comment character	`%`
15	Invalid character	⟨delete⟩

When `INITEX` begins, category 12 (other) has been assigned to all 256 possible characters, except that the 52 letters `A...Z` and `a...z` are category 11 (letter), and assignments equivalent to the following have been made:

```
\catcode `\\ = 0
\catcode `\^^M = 5
\catcode `\^^@ = 9
\catcode `\  = 10
\catcode `\% = 14
\catcode `\^^? = 15
```

Thus '\' is already an escape character, '`\char"20`' is a space, and '`%`' is available for comments on the first line of the file; ASCII ⟨null⟩ is ignored, ASCII ⟨return⟩ is an end-of-line character, and ASCII ⟨delete⟩ is invalid.

Furthermore ⟨tab⟩ is given category space, ⟨formfeed⟩ becomes an active character that will detect runaways on files that have been divided into "file pages" by ⟨formfeed⟩ characters. Finally the control sequence `\active` is defined to yield the constant 13.

To re-catcode these special characters—not counting ASCII

⟨null⟩	`^^@`
⟨tab⟩	`^^I`
⟨linefeed⟩	`^^J`
⟨formfeed⟩	`^^L`
⟨return⟩	`^^M`
⟨delete⟩	`^^?`

—use the control sequence `\dospecials` that lists all the characters whose catcodes should probably be changed to 12 (other) when copying things verbatim. Each symbol in the list is preceded by `\do`, which can be defined if you want to do something to every item in the list.

⟨ ***Establish standard category code values*** ⟩≡

```
\catcode`\{=1
\catcode`\}=2
\catcode`\$=3
\catcode`\&=4
\catcode`\#=6
\catcode`\^=7 \catcode`\^^K=7 % uparrow is for superscripts
\catcode`\_=8 \catcode`\^^A=8 % downarrow are for subscripts
\catcode`\^^I=10

\chardef\active=13 \catcode`\~=\active % tilde is active
\catcode`\^^L=\active \outer\def^^L{\par} % ascii form-feed is "\outer\par"

\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\^^K\do\_\do\^^A\do\%\do\~}
```

To make the plain macros more efficient in time and space, several constant values are declared as control sequences. *If they were changed, anything could happen.* So be careful!

⟨ ***Define commonly used constants*** ⟩≡

```
\chardef\@ne=1
\chardef\tw@=2
\chardef\thr@@=3
\chardef\sixt@@n=16
\chardef\@cclv=255
\mathchardef\@cclvi=256
\mathchardef\@m=1000
\mathchardef\@M=10000
\mathchardef\@MM=20000
```

Text fonts

⟨ *Set up text fonts* ⟩≡

⟨ *Provide support for font scaling* ⟩
⟨ *Define text fonts* ⟩
⟨ *Encode special characters, and characters not available on the keyboard* ⟩
⟨ *Provide support for accented characters* ⟩
⟨ *Assign uppercase and lowercase code values* ⟩
⟨ *Assign space factor codes* ⟩

Fonts assigned to `\preloaded` are not part of the format, but they are preloaded so that other format packages can use them. For example, if another set of macros says `\font\ninerm=cmr9`, TEX will not have to reload the font metric information for `cmr9`.

⟨ *Define text fonts* ⟩≡

```
\font\tenrm=cmr10 % roman text
\font\preloaded=cmr9
\font\preloaded=cmr8
\font\sevenrm=cmr7
\font\preloaded=cmr6
\font\fiverm=cmr5

\font\preloaded=cmss10 % sans serif
\font\preloaded=cmssq8

\font\preloaded=cmssi10 % sans serif italic
\font\preloaded=cmssqi8

\font\tenbf=cmbx10 % boldface extended
\font\preloaded=cmbx9
\font\preloaded=cmbx8
\font\sevenbf=cmbx7
\font\preloaded=cmbx6
\font\fivebf=cmbx5

\font\tentt=cmtt10 % typewriter
\font\preloaded=cmtt9
\font\preloaded=cmtt8

\font\preloaded=cmsltt10 % slanted typewriter

\font\tensl=cmsl10 % slanted roman
\font\preloaded=cmsl9
\font\preloaded=cmsl8

\font\tenit=cmti10 % text italic
\font\preloaded=cmti9
\font\preloaded=cmti8
\font\preloaded=cmti7

\font\preloaded=cmu10 % unslanted text italic

```

```
\font\preloaded=cmcsc10 % caps and small caps

\font\preloaded=cmssbx10 % sans serif bold extended

\font\preloaded=cmdunh10 % Dunhill style

\font\preloaded=cmr7 scaled \magstep4 % for titles
\font\preloaded=cmtt10 scaled \magstep2
\font\preloaded=cmssbx10 scaled \magstep2

\font\preloaded=manfnt % METAFONT logo and dragon curve and special symbols
```

Additional `\preloaded` fonts can be specified here. (And those that were `\preloaded` above can be eliminated.)

⟨ ***Define text fonts*** ⟩+≡

```
\let\preloaded=\undefined % preloaded fonts must be declared anew later.
```

⟨ ***Provide support for font scaling*** ⟩≡

```
\def\magstephalf{1095 }
\def\magstep#1{\ifcase#1 \@m\or 1200\or 1440\or 1728\or 2074\or 2488\fi\relax}
\def\magnification{\afterassignment\m@g\count@}
\def\m@g{\mag\count@
  \hsize6.5truein\vsize8.9truein\dimen\footins8truein}
```

Font encoding

We usually think of text files as containing characters. It doesn't cause any problems most of the time when we use plain ASCII characters—letters A–Z, a–z, the numerals 0–9 and some of punctuation characters. This illusion is broken down when we start using characters that do not belong to this limited set, for example, accented characters / mathematical symbols. Then what we see on screen may not match what we key in. What gets printed may not match what we see on screen. Moreover, what gets shown on screen and what gets printed depends on what machine we are on and how the fonts that we are using are set up. In reality text files contain just numeric codes (in range 0–255) stored in 8-bit bytes, and the mapping between 'character' and numeric code is quite arbitrary. This is because there are very many more characters than the 256 numeric codes possible with 8-bits. Consequently, there will be a need for more than one possible mapping or 'font encoding', or in other words, there would not be a 'standard' encoding that suits all purposes.

When a symbol is built up by forming a box, the `\leavevmode` macro is called first; this starts a new paragraph, if TEX is in vertical mode, but does nothing if TEX is in horizontal mode or math mode. `\chardef` positions are taken from the fonts `cmr10` and `cmsy10`.

⟨ ***Encode special characters, and characters not available on the keyboard*** ⟩≡

```
\chardef\%='\%
\chardef\&='\&
\chardef\#='\#
\chardef\$='\$
\chardef\ss="19
\chardef\ae="1A
\chardef\oe="1B
\chardef\o="1C
```

```
\chardef\AE="1D
\chardef\OE="1E
\chardef\O="1F
\chardef\i="10 \chardef\j="11 % dotless letters
\def\aa{\accent23a}
\def\l{\char321}
\def\L{\leavevmode\setbox0\hbox{L}\hbox to\wd0{\hss\char32L}}

\def\leavevmode{\unhbox\voidb@x} % begins a paragraph, if necessary
\def\_{\leavevmode \kern.06em \vbox{\hrule width.3em}}
\def\AA{\leavevmode\setbox0\hbox{h}\dimen@\ht0\advance\dimen@-1ex%
  \rlap{\raise.67\dimen@\hbox{\char'27}}A}

\def\mathhexbox#1#2#3{\leavevmode
  \hbox{$\m@th \mathchar"#1#2#3$}}
\def\dag{\mathhexbox279}
\def\ddag{\mathhexbox27A}
\def\S{\mathhexbox278}
\def\P{\mathhexbox27B}
```

The accent positions are taken from Computer Modern font family. We are about to 'hard-wire' CM accent encoding into the format. Different encoding will be necessary if other styles of type are used.

Three alternative control-symbol accents are defined, suitable for keyboards with extended character sets: `\let\^^_=\v`, `\let\^^S=\u`, `\let\^^D=\^`.

⟨ ***Provide support for accented characters*** ⟩≡

```
\def\oalign#1{\leavevmode\vtop{\baselineskip\z@skip \lineskip.25ex%
  \ialign{##\crcr#1\crcr}}} \def\o@lign{\lineskiplimit\z@ \oalign}
\def\ooalign{\lineskiplimit-\maxdimen \oalign} % chars over each other
\def\sh@ft#1{\dimen\z@.00#1ex\multiply\dimen\z@\fontdimen1\font
  \kern-.0156\dimen\z@} % compensate for slant in lowered accents
\def\d#1{{\o@lign{\relax#1\crcr\hidewidth\sh@ft{10}.\hidewidth}}}
\def\b#1{{\o@lign{\relax#1\crcr\hidewidth\sh@ft{29}%
    \vbox to.2ex{\hbox{\char22}\vss}\hidewidth}}}
\def\c#1{\setbox\z@\hbox{#1}\ifdim\ht\z@=1ex\accent24 #1%
  \else{\ooalign{\unhbox\z@\crcr\hidewidth\char24\hidewidth}}\fi}
\def\copyright{{\ooalign{\hfil\raise.07ex\hbox{c}\hfil\crcr\mathhexbox20D}}}

\def\dots{\relax\ifmmode\ldots\else$\m@th\ldots\,$\fi}
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX}

\def\‘#1{{\accent18 #1}}
\def\’#1{{\accent19 #1}}
\def\v#1{{\accent20 #1}} \let\^^_=\v
\def\u#1{{\accent21 #1}} \let\^^S=\u
\def\=#1{{\accent22 #1}}
\def\^#1{{\accent94 #1}} \let\^^D=\^
\def\.#1{{\accent95 #1}}
\def\H#1{{\accent"7D #1}}
\def\~#1{{\accent"7E #1}}
\def\"#1{{\accent"7F #1}}
\def\t#1{{\edef\next{\the\font}\the\textfont1\accent"7F\next#1}}
```

INITEX sets \uccode‘x = ‘X and \uccode‘X = ‘X for all letters x, and \lccode‘x = ‘x, \lccode‘X = ‘x; all other values are zero.

⟨ ***Assign uppercase and lowercase code values*** ⟩≡

```
133 %%% Thats all for English language.
```

Space factor code affects setting of interword glue. The space factor is normally 1000, which means that the interword glue should not be modified. If the space factor f is different from 1000, the interword glue is computed as follows: Take the normal space glue for the current font, and add the extra space if $f \geq 2000$. Then the stretch component is multiplied by $f/1000$, while the shrink component is multiplied by $1000/f$. (Look up the Appendix for the values of normal space, normal stretch, normal shrink, and extra space for some of CM fonts.)

INITEX sets space factor codes: \sfcode $x = 1000$ for all x, except that \sfcode‘$X = 999$ for uppercase letters.

The characters ‘)’, ‘’’, and ‘]’ does not change space factor.

⟨ ***Assign space factor codes*** ⟩≡

```
134 \sfcode‘\)=0 \sfcode‘\’=0 \sfcode‘\]=0
```

Math fonts

⟨ ***Set up math fonts*** ⟩≡

⟨ ***Define math fonts*** ⟩
⟨ ***Encode math accents*** ⟩
⟨ ***Establish spacing around mathematical objects*** ⟩
⟨ ***Assign math codes*** ⟩
⟨ ***Assign delimiter codes*** ⟩
⟨ ***Define font families*** ⟩

As was said earlier, the font metric information about preloaded font will be build into the format. But, if another set of macros says \font\fiftyfiverm = cmr9 at 55pt, TEX will *have to reload again* the font metric information for cmr9.

⟨ ***Define math fonts*** ⟩≡

```
135 \font\teni=cmmi10 % math italic
136 \font\preloaded=cmmi9
137 \font\preloaded=cmmi8
138 \font\seveni=cmmi7
139 \font\preloaded=cmmi6
140 \font\fivei=cmmi5
141
142 \font\tensy=cmsy10 % math symbols
143 \font\preloaded=cmsy9
144 \font\preloaded=cmsy8
145 \font\sevensy=cmsy7
146 \font\preloaded=cmsy6
147 \font\fivesy=cmsy5
148
149 \font\tenex=cmex10 % math extension
150
```

```
\font\preloaded=cmmib10 % bold math italic
\font\preloaded=cmbsy10 % bold math symbols
```

Mathematical spacing

Spacing around mathematical object is measured in `mu`—'math units.' `1mu` is equal to 1/18 th part of `\fontdimen 6` of the font in family 2.

`\quad` spacing does not change with the style of formula, nor does it depend on the math font families that are being used. But thin spaces, medium spaces, and thick spaces do get bigger and smaller as the size of type gets bigger and smaller; this is because they are defined in terms of ⟨muglue⟩.

According to these specifications, thin spaces in plain TEX do not stretch or shrink; medium spaces can stretch a little, and they can shrink to zero; thick spaces can stretch a lot, but they never shrink.

The following table gives the complete definition of muglue between mathematical objects. A formula is converted to a math list, and the math list consists chiefly of "atoms" of eight basic types: Ord (ordinary), Op (large operator), Bin (binary operation), Rel (relation), Open (opening), Close (closing), Punct (punctuation), and Inner (a delimited subformula). Other kinds of atoms, which arise from commands like `\overline` or `\mathaccent` or `\vcenter`, etc., are all treated as type Ord; fractions are treated as type Inner. The following (non-symmetric) table is used to determine the spacing between pairs of adjacent atoms:

		Right atom							
		Ord	Op	Bin	Rel	Open	Close	Punct	Inner
	Ord	0	1	(2)	(3)	0	0	0	(1)
	Op	1	1	*	(3)	0	0	0	(1)
	Bin	(2)	(2)	*	*	(2)	*	*	(2)
Left	Rel	(3)	(3)	*	0	(3)	0	0	(3)
atom	Open	0	0	*	0	0	0	0	0
	Close	0	1	(2)	(3)	0	0	0	(1)
	Punct	(1)	(1)	*	(1)	(1)	(1)	(1)	(1)
	Inner	(1)	1	(2)	(3)	(1)	0	(1)	(1)

Here `0`, `1`, `2`, and `3` stand for no space, thin space, medium space, and thick space, respectively. Thin space, medium space, and thin space are equal to values of `\thinmuskim`, `\medmuskip`, `\thick-muskip` parameters, respectively. The table entry is parenthesized if the space is to be inserted only in display and text styles, not in script and scriptscript styles. For example, many of the entries in the Rel row and the Rel column are '`(3)`'; this means that thick spaces are normally inserted before and after relational symbols like '=', but not in subscripts. Some of the entries in the table are '`*`'; such cases never arise, because Bin atoms must be preceded and followed by atoms compatible with the nature of binary operations. The conversion of math lists to horizontal lists is done whenever TEX is about to leave math mode, and the inter-atomic spacing is inserted at that time.

⟨ ***Establish spacing around mathematical objects*** ⟩≡

```
\thinmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
```

For the positioning of accents over single character the width of `\skewchar` is used. For most of fonts the default value of `\skewchar` is `-1`; but the math italic (family 1) and math symbol fonts (family 2) have special `\skewchar` values equal to `'177` and `'60`, respectively. These are characters '⁀' and '⁄'.

⟨ ***Encode math accents*** ⟩≡

```
\skewchar\teni='177 \skewchar\seveni='177 \skewchar\fivei='177
\skewchar\tensy='60 \skewchar\sevensy='60 \skewchar\fivesy='60
```

A math code is relevant only when the corresponding category code is 11 or 12. When processing in math mode characters of categories 11 and 12, `\char` and `\chardef` characters are replaced by their math code.

If we denote 15-bit number by `"uvwz`, then math codes are assigned by

`\mathcode` ⟨8-bit number⟩ = `"uvwz`, where

`u` — the class code (see below for the list)
`v` — the font family number (see the font tables at the end of this document)
`wz` — the position of the character in the font

Class	*Meaning*	*Example*	*Class*	*Meaning*	*Example*
0	Ordinary	`/`	4	Opening	`(`
1	Large operator	`\sum`	5	Closing	`)`
2	Binary operation	`+`	6	Punctuation	`,`
3	Relation	`=`	7	Variable family	`x`

A `\mathcode` can also have the special value `"8000`, which causes the character to behave as if it has catcode 13 (active). This feature makes `'` apostrophe expand to `\prime`. The mathcode of `'` does not interfere with the use of `'` in octal constants. The mathcode of `"8000` is also assigned to space and underscore.

`INITEX` sets up `\mathcode` $x = x$ for $x = 0..255$, except that `\mathcode` $x = x +$ `"7000` for each of the ten digits $x =$ `` `0 `` to `` `9 ``; `\mathcode` $x = x +$ `"7100` for each of the 52 letters. TEX looks at the mathcode only when it is typesetting a character whose catcode is 11 (letter) or 12 (other), or when it encounters a character that is given explicitly as `\char`⟨number⟩.

Class 7 is a special case that allows math symbols to change families. It behaves exactly like class 0, except that the specified family is replaced by the current value of an integer parameter called `\fam`, provided that `\fam` is a legal family number (i.e., if it lies between 0 and 15). TEX automatically sets `\fam=-1` whenever math mode is entered; therefore class 7 and class 0 are equivalent unless `\fam` has been given a new value. Plain TEX changes `\fam` to 0 when the user types '`\rm`'; this makes it convenient to get roman letters in formulas, since letters belong to class 7. The control sequence `\rm` is an abbreviation for '`\fam=0 \tenrm`'; thus, `\rm` causes `\fam` to become zero, and it makes `\tenrm` the "current font." In horizontal mode, the `\fam` value is irrelevant and the current font governs the typesetting of letters; but in math mode, the current font is irrelevant and the `\fam` value governs the letters. The current font affects math mode only if control space (`\ `) is used or if dimensions are given in `ex` or `em` units; it also has an effect if an `\hbox` appears inside a formula, since the contents of an hbox are typeset in horizontal mode.

⟨ ***Assign math codes*** ⟩≡

```
\mathcode`\^^@="2201 % \cdot
\mathcode`\^^A="3223 % \downarrow
\mathcode`\^^B="010B % \alpha
\mathcode`\^^C="010C % \beta
\mathcode`\^^D="225E % \land
\mathcode`\^^E="023A % \lnot
```

```
\mathcode`\^^F="3232 % \in
\mathcode`\^^G="0119 % \pi
\mathcode`\^^H="0115 % \lambda
\mathcode`\^^I="010D % \gamma
\mathcode`\^^J="010E % \delta
\mathcode`\^^K="3222 % \uparrow
\mathcode`\^^L="2206 % \pm
\mathcode`\^^M="2208 % \oplus
\mathcode`\^^N="0231 % \infty
\mathcode`\^^O="0140 % \partial
\mathcode`\^^P="321A % \subset
\mathcode`\^^Q="321B % \supset
\mathcode`\^^R="225C % \cap
\mathcode`\^^S="225B % \cup
\mathcode`\^^T="0238 % \forall
\mathcode`\^^U="0239 % \exists
\mathcode`\^^V="220A % \otimes
\mathcode`\^^W="3224 % \leftrightarrow
\mathcode`\^^X="3220 % \leftarrow
\mathcode`\^^Y="3221 % \rightarrow
\mathcode`\^^Z="8000 % \ne
\mathcode`\^^[="2205 % \diamond
\mathcode`\^^\="3214 % \le
\mathcode`\^^]="3215 % \ge
\mathcode`\^^^="3211 % \equiv
\mathcode`\^^_="225F % \lor
\mathcode`\ ="8000 % \space
\mathcode`\!="5021
\mathcode`\'="8000 % ^\prime
\mathcode`\(="4028
\mathcode`\)="5029
\mathcode`\*="2203 % \ast
\mathcode`\+="202B
\mathcode`\,="613B
\mathcode`\-="2200
\mathcode`\.="013A
\mathcode`\/="013D
\mathcode`\:="303A
\mathcode`\;="603B
\mathcode`\<="313C
\mathcode`\=="303D
\mathcode`\>="313E
\mathcode`\?="503F
\mathcode`\[="405B
\mathcode`\\="026E % \backslash
\mathcode`\]="505D
\mathcode`\_="8000 % \_
\mathcode`\{="4266
\mathcode`\|="026A
\mathcode`\}="5267
\mathcode`\^^?="1273 % \smallint
```

Delimiter codes are used after `\left` and `\right` commands, when TEX is looking for a delimiter. If we denote 24-bit number by `"qrstuv`, then delimiter codes are assigned by

`\delcode` ⟨8-bit number⟩ = `"qrstuv`, where

`q` — the font family number of
`rs` — the position of the the small variant of the delimiter
`t` — the font family number of
`uv` — the position of the the large variant of the delimiter

`INITEX` sets all `\delcode` values to `-1`, which means that no characters are recognized as delimiters in math formulas, except `\delcode`‘. = 0, so that ‘.’ stands for “null delimiter”. `{` and `}` should *not get* delcodes; otherwise parameter grouping fails!

⟨ ***Assign delimiter codes*** ⟩≡

```
\delcode‘\(="028300
\delcode‘\)="029301
\delcode‘\[="05B302
\delcode‘\]="05D303
\delcode‘\<="26830A
\delcode‘\>="26930B
\delcode‘\/="02F30E
\delcode‘\|="26A30C
\delcode‘\\="26E30F
```

All characters that are typeset in math mode belong to one of sixteen *families* of fonts, numbered internally from 0 to 15. Each of these families consists of three fonts: one for text size, one for script size, and one for scriptscriptsize. The commands `\textfont`, `\scriptfont`, and `\scriptscriptfont` are used to specify the members of each family. Since there are up to 256 characters per font, and 3 fonts per family, and 16 families, TEX can access up to 12,288 characters in any one formula (4096 in each of the three sizes).

The `plain.tex` format uses family 1 for math italic letters, family 2 for ordinary math symbols, and family 3 for large symbols. Text italic is put in family 4, slanted roman in family 5, bold roman in family 6, and typewriter type in family 7. A macro `\newfam` will assign symbolic names to families that aren’t already used.

`INITEX` initializes the mathcodes of all letters `A` to `Z` and `a` to `z` so that they are symbols of class 7 and family 1; that’s why it is natural to use family 1 for math italics. Similarly, the digits `0` to `9` are class 7 and family 0. None of the other families is treated in any special way by TEX.

TEX doesn’t check to see if the families are sensibly organized. The only constraint is that the fonts in families 2 and 3 have special `\fontdimen` parameters, which govern mathematical spacing (see Appendix). In Computer Modern only `cmsy` and `cmex` have these parameters, so their assignment to families 2 and 3 is almost mandatory.

During the time that a math formula is being read, TEX remembers each symbol as being “character position so-and-so in family number such-and-such,” but it does not take note of what fonts are actually in the families until reaching the end of the formula.

⟨ ***Define font families*** ⟩≡

```
\textfont0=\tenrm \scriptfont0=\sevenrm \scriptscriptfont0=\fiverm
\def\rm{\fam\z@\tenrm}
\textfont1=\teni \scriptfont1=\seveni \scriptscriptfont1=\fivei
\def\mit{\fam\@ne} \def\oldstyle{\fam\@ne\teni}
\textfont2=\tensy \scriptfont2=\sevensy \scriptscriptfont2=\fivesy
\def\cal{\fam\tw@}
\textfont3=\tenex \scriptfont3=\tenex \scriptscriptfont3=\tenex
```

```
\newfam\itfam \def\it{\fam\itfam\tenit} % \it is family 4
\textfont\itfam=\tenit
\newfam\slfam \def\sl{\fam\slfam\tensl} % \sl is family 5
\textfont\slfam=\tensl
\newfam\bffam \def\bf{\fam\bffam\tenbf} % \bf is family 6
\textfont\bffam=\tenbf \scriptfont\bffam=\sevenbf
\scriptscriptfont\bffam=\fivebf
\newfam\ttfam \def\tt{\fam\ttfam\tentt} % \tt is family 7
\textfont\ttfam=\tentt
```

Registers allocation

Here are macros for the automatic allocation of `\count`, `\box`, `\dimen`, `\skip`, `\muskip`, and `\toks` registers, as well as `\read` and `\write` stream numbers, `\fam` codes, `\language` codes, and `\insert` numbers.

The main use of these macros is for registers that are defined by one macro and used by others, possibly at different nesting levels.

The following counters are reserved:

0–9	page numbering
10	count allocation
11	dimen allocation
12	skip allocation
13	muskip allocation
14	box allocation
15	toks allocation
16	read file allocation
17	write file allocation
18	math family allocation
19	language allocation
20	insert allocation
21	the most recently allocated number
22	constant -1

New counters are allocated starting with 23, 24, etc. Other registers are allocated starting with 10. This leaves 0 through 9 for the user to play with safely, except that counts 0 to 9 are considered to be the page and subpage numbers (since they are displayed during output). In this scheme, `\count` 10 always contains the number of the highest-numbered counter that has been allocated, `\count 14` the highest-numbered box, etc. Inserts are given numbers 254, 253, etc., since they require a `\count`, `\dimen`, `\skip`, and `\box` all with the same number; `\count` 20 contains the lowest-numbered insert that has been allocated. `\box255` is reserved for `\output`; `\count255`, `\dimen255`, and `\skip255` can be used freely.

It is recommended that macro designers always use global assignments with respect to registers numbered 1, 3, 5, 7, 9, and always non-global assignments with respect to registers 0, 2, 4, 6, 8, 255. This will prevent "save stack buildup" that might otherwise occur.

⟨ ***Allocate registers*** ⟩≡

```
\count10=22 % allocates \count registers 23, 24, ...
\count11=9 % allocates \dimen registers 10, 11, ...
\count12=9 % allocates \skip registers 10, 11, ...
\count13=9 % allocates \muskip registers 10, 11, ...
\count14=9 % allocates \box registers 10, 11, ...
```

```
\count15=9 % allocates \toks registers 10, 11, ...
\count16=-1 % allocates input streams 0, 1, ...
\count17=-1 % allocates output streams 0, 1, ...
\count18=3 % allocates math families 4, 5, ...
\count19=0 % allocates \language codes 1, 2, ...
\count20=255 % allocates insertions 254, 253, ...
\countdef\insc@unt=20 % the insertion counter
\countdef\allocationnumber=21 % the most recent allocation
\countdef\m@ne=22 \m@ne=-1 % a handy constant
\def\wlog{\immediate\write\m@ne} % write on log file (only)
⟨ Allocate scratch registers ⟩
⟨ Provide user-level register allocation macros ⟩
⟨ Define implementation-level register allocation macros ⟩
⟨ Initialize register constants ⟩
```

Here are abbreviations for the names of scratch registers that don't need to be allocated.

⟨ *Allocate scratch registers* ⟩≡

```
\countdef\count@=255
\dimendef\dimen@=0
\dimendef\dimen@i=1 % global only
\dimendef\dimen@ii=2
\skipdef\skip@=0
\toksdef\toks@=0
```

Now, we define `\newcount`, `\newbox`, etc. so that you can say `\newcount\foo` and `\foo` will be defined (with `\countdef`) to be the next counter. To find out which counter `\foo` is, you can look at `\allocationnumber`. Since there's no `\boxdef` command, `\chardef` is used to define a `\newbox`, `\newinsert`, `\newfam`, and so on.

⟨ *Provide user-level register allocation macros* ⟩≡

```
\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\outer\def\newskip{\alloc@2\skip\skipdef\insc@unt}
\outer\def\newmuskip{\alloc@3\muskip\muskipdef\@cclvi}
\outer\def\newbox{\alloc@4\box\chardef\insc@unt}
\let\newtoks=\relax % we do this to allow plain.tex to be read in twice
\outer\def\newhelp#1#2{\newtoks#1#1\expandafter{\csname#2\endcsname}}
\outer\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
\outer\def\newread{\alloc@6\read\chardef\sixt@@n}
\outer\def\newwrite{\alloc@7\write\chardef\sixt@@n}
\outer\def\newfam{\alloc@8\fam\chardef\sixt@@n}
\outer\def\newlanguage{\alloc@9\language\chardef\@cclvi}
```

⟨ *Define implementation-level register allocation macros* ⟩≡

```
\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
  \ch@ck#1#4#2% make sure there's still room
  \allocationnumber=\count1#1%
  \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}
\outer\def\newinsert#1{\global\advance\insc@unt by\m@ne
  \ch@ck0\insc@unt\count
```

```
    \ch@ck1\insc@unt\dimen
    \ch@ck2\insc@unt\skip
    \ch@ck4\insc@unt\box
    \allocationnumber=\insc@unt
    \global\chardef#1=\allocationnumber
    \wlog{\string#1=\string\insert\the\allocationnumber}}
\def\ch@ck#1#2#3{\ifnum\count1#1<#2%
    \else\errmessage{No room for a new #3}\fi}
```

We finish with the initalization of some constants.

⟨ ***Initialize register constants*** ⟩≡

```
\newdimen\maxdimen \maxdimen=16383.99999pt % the largest legal <dimen>
\newskip\hideskip \hideskip=-1000pt plus 1fill % negative but can grow
\newskip\centering \centering=0pt plus 1000pt minus 1000pt
\newdimen\p@ \p@=1pt % this saves macro space and time
\newdimen\z@ \z@=0pt % can be used both for 0pt and 0
\newskip\z@skip \z@skip=0pt plus0pt minus0pt
\newbox\voidb@x % permanently void box register
```

Parameters

Let's turn now to TEX's parameters, which the previous chapters have introduced one at a time; it will be convenient to assemble them all together.

An ⟨integer parameter⟩ is one of the following tokens:

`\pretolerance` badness tolerance before hyphenation
`\tolerance` badness tolerance after hyphenation
`\hbadness` badness above which bad hboxes will be shown
`\vbadness` badness above which bad vboxes will be shown
`\linepenalty` amount added to badness of every line in a paragraph
`\hyphenpenalty` penalty for line break after discretionary hyphen
`\exhyphenpenalty` penalty for line break after explicit hyphen
`\binoppenalty` penalty for line break after binary operation
`\relpenalty` penalty for line break after math relation
`\clubpenalty` penalty for creating a club line at bottom of page
`\widowpenalty` penalty for creating a widow line at top of page
`\displaywidowpenalty` ditto, before a display
`\brokenpenalty` penalty for page break after a hyphenated line
`\predisplaypenalty` penalty for page break just before a display
`\postdisplaypenalty` penalty for page break just after a display
`\interlinepenalty` additional penalty for page break between lines
`\floatingpenalty` penalty for insertions that are split
`\outputpenalty` penalty at the current page break
`\doublehyphendemerits` demerits for consecutive broken lines
`\finalhyphendemerits` demerits for a penultimate broken line
`\adjdemerits` demerits for adjacent incompatible lines
`\looseness` change to the number of lines in a paragraph
`\pausing` positive if pausing after each line is read from a file
`\holdinginserts` positive if insertions remain dormant in output box
`\tracingonline` positive if showing diagnostic info on the terminal
`\tracingmacros` positive if showing macros as they are expanded

\tracingstats positive if showing statistics about memory usage
\tracingparagraphs positive if showing line-break calculations
\tracingpages positive if showing page-break calculations
\tracingoutput positive if showing boxes that are shipped out
\tracinglostchars positive if showing characters not in the font
\tracingcommands positive if showing commands before they are executed
\tracingrestores positive if showing deassignments when groups end
\language the current set of hyphenation rules
\uchyph positive if hyphenating words beginning with capital letters
\lefthyphenmin smallest fragment at beginning of hyphenated word
\righthyphenmin smallest fragment at end of hyphenated word
\globaldefs nonzero if overriding \global specifications
\defaulthyphenchar \hyphenchar value when a font is loaded
\defaultskewchar \skewchar value when a font is loaded
\escapechar escape character in the output of control sequence tokens
\endlinechar character placed at the right end of an input line
\newlinechar character that starts a new output line
\maxdeadcycles upper bound on \deadcycles
\hangafter hanging indentation changes after this many lines
\fam the current family number
\mag magnification ratio, times 1000
\delimiterfactor ratio for variable delimiters, times 1000
\time current time of day in minutes since midnight
\day current day of the month
\month current month of the year
\year current year of our Lord
\showboxbreadth maximum items per level when boxes are shown
\showboxdepth maximum level when boxes are shown
\errorcontextlines maximum extra context shown when errors occur

The first few of these parameters have values in units of "badness" and "penalties" that affect line breaking and page breaking. Then come demerit-oriented parameters; demerits are essentially given in units of "badness squared," so those parameters tend to have larger values. By contrast, the next few parameters (\looseness, \pausing, etc.) generally have quite small values (either -1 or 0 or 1 or 2). Miscellaneous parameters complete the set.

A ⟨dimen parameter⟩ is one of the following:

\hfuzz maximum overrun before overfull hbox messages occur
\vfuzz maximum overrun before overfull vbox messages occur
\overfullrule width of rules appended to overfull boxes
\emergencystretch reduces badnesses on final pass of line-breaking
\hsize line width in horizontal mode
\vsize page height in vertical mode
\maxdepth maximum depth of boxes on main pages
\splitmaxdepth maximum depth of boxes on split pages
\boxmaxdepth maximum depth of boxes on explicit pages
\lineskiplimit threshold where \baselineskip changes to \lineskip
\delimitershortfall maximum space not covered by a delimiter
\nulldelimiterspace width of a null delimiter
\scriptspace extra space after subscript or superscript
\mathsurround kerning before and after math in text
\predisplaysize length of text preceding a display
\displaywidth length of line for displayed equation

`\displayindent` indentation of line for displayed equation
`\parindent` width of `\indent`
`\hangindent` amount of hanging indentation
`\hoffset` horizontal offset in `\shipout`
`\voffset` vertical offset in `\shipout`

Before typesetting a delimiter TEX determines the size f of the formula to be covered as twice the maximum of the height and the depth of the formula. The size d of the delimiter should be

$$\min\bigl\{f - \texttt{\delimitershortfall},\ \texttt{\delimiterfactor}\cdot\frac{f}{1000}\bigr\} \le d$$

And the possibilities for ⟨glue parameter⟩ are:

`\baselineskip` desired glue between baselines
`\lineskip` interline glue if `\baselineskip` isn't feasible
`\parskip` extra glue just above paragraphs
`\abovedisplayskip` extra glue just above displays
`\abovedisplayshortskip` ditto, following short lines
`\belowdisplayskip` extra glue just below displays
`\belowdisplayshortskip` ditto, following short lines
`\leftskip` glue at left of justified lines
`\rightskip` glue at right of justified lines
`\topskip` glue at top of main pages
`\splittopskip` glue at top of split pages
`\tabskip` glue between aligned entries
`\spaceskip` glue between words, if nonzero
`\xspaceskip` glue between sentences, if nonzero
`\parfillskip` additional `\rightskip` at end of paragraphs

To above parameters (except `\parfillskip`) are assigned values appropriate for CM family typeset at 12 pt baseline.

Finally, there are three permissible ⟨muglue parameter⟩ tokens:

`\thinmuskip` thin space in math formulas
`\medmuskip` medium space in math formulas
`\thickmuskip` thick space in math formulas

TEX also has parameters that are token lists. Such parameters do not enter into the definitions of ⟨number⟩ and such things. A ⟨token parameter⟩ is any of:

`\output` the user's output routine
`\everypar` tokens to insert when a paragraph begins
`\everymath` tokens to insert when math in text begins
`\everydisplay` tokens to insert when display math begins
`\everyhbox` tokens to insert when an hbox begins
`\everyvbox` tokens to insert when a vbox begins
`\everyjob` tokens to insert when the job begins
`\everycr` tokens to insert after every `\cr` or nonredundant `\crcr`
`\errhelp` tokens that supplement an `\errmessage`

All of numeric parameters are listed below, but the code is commented out if no special value needs to be set. `INITEX` makes all parameters zero except where noted.

⟨ *Assign initial values to parameters* ⟩≡

⟨ *Assign values to integer parameters* ⟩
⟨ *Assign values to dimen parameters* ⟩
⟨ *Assign values to glue parameters* ⟩
⟨ *Assign values to special registers* ⟩

⟨ *Assign values to integer parameters* ⟩≡

```
\pretolerance=100
\tolerance=200 % INITEX sets this to 10000
\hbadness=1000
\vbadness=1000
\linepenalty=10
\hyphenpenalty=50
\exhyphenpenalty=50
\binoppenalty=700
\relpenalty=500
\clubpenalty=150
\widowpenalty=150
\displaywidowpenalty=50
\brokenpenalty=100
\predisplaypenalty=10000
% \postdisplaypenalty=0
% \interlinepenalty=0
% \floatingpenalty=0, set during \insert
% \outputpenalty=0, set before TeX enters \output
\doublehyphendemerits=10000
\finalhyphendemerits=5000
\adjdemerits=10000
% \looseness=0, cleared by TeX after each paragraph
% \pausing=0
% \holdinginserts=0
% \tracingonline=0
% \tracingmacros=0
% \tracingstats=0
% \tracingparagraphs=0
% \tracingpages=0
% \tracingoutput=0
\tracinglostchars=1
% \tracingcommands=0
% \tracingrestores=0
% \language=0
\uchyph=1
% \lefthyphenmin=2 \righthyphenmin=3 set below
% \globaldefs=0
% \maxdeadcycles=25 % INITEX does this
% \hangafter=1 % INITEX does this, also TeX after each paragraph
% \fam=0
% \mag=1000 % INITEX does this
% \escapechar=‘\\ % INITEX does this
\defaulthyphenchar=‘\-
\defaultskewchar=-1
% \endlinechar=‘\^^M % INITEX does this
```

```
\newlinechar=-1
\delimiterfactor=901
% \time=now % TeX does this at beginning of job
% \day=now % TeX does this at beginning of job
% \month=now % TeX does this at beginning of job
% \year=now % TeX does this at beginning of job
\showboxbreadth=5
\showboxdepth=3
\errorcontextlines=5
```

⟨ ***Assign values to dimen parameters*** ⟩≡

```
\hfuzz=0.1pt
\vfuzz=0.1pt
\overfullrule=5pt
\hsize=6.5in
\vsize=8.9in
\maxdepth=4pt
\splitmaxdepth=\maxdimen
\boxmaxdepth=\maxdimen
% \lineskiplimit=0pt, changed by \normalbaselines
\delimitershortfall=5pt
\nulldelimiterspace=1.2pt
\scriptspace=0.5pt
% \mathsurround=0pt
% \predisplaysize=0pt, set before TeX enters $$
% \displaywidth=0pt, set before TeX enters $$
% \displayindent=0pt, set before TeX enters $$
\parindent=20pt
% \hangindent=0pt, zeroed by TeX after each paragraph
% \hoffset=0pt
% \voffset=0pt
```

⟨ ***Assign values to glue parameters*** ⟩≡

```
% \baselineskip=0pt, changed by \normalbaselines
% \lineskip=0pt, changed by \normalbaselines
\parskip=0pt plus 1pt
\abovedisplayskip=12pt plus 3pt minus 9pt
\abovedisplayshortskip=0pt plus 3pt
\belowdisplayskip=12pt plus 3pt minus 9pt
\belowdisplayshortskip=7pt plus 3pt minus 4pt
% \leftskip=0pt
% \rightskip=0pt
\topskip=10pt
\splittopskip=10pt
% \tabskip=0pt
% \spaceskip=0pt
% \xspaceskip=0pt
\parfillskip=0pt plus 1fil
```

We also define special registers that function like parameters:

⟨ ***Assign values to special registers*** ⟩≡

```
\newskip\smallskipamount \smallskipamount=3pt plus 1pt minus 1pt
\newskip\medskipamount \medskipamount=6pt plus 2pt minus 2pt
\newskip\bigskipamount \bigskipamount=12pt plus 4pt minus 4pt
\newskip\normalbaselineskip \normalbaselineskip=12pt
\newskip\normallineskip \normallineskip=1pt
\newdimen\normallineskiplimit \normallineskiplimit=0pt
\newdimen\jot \jot=3pt
\newcount\interdisplaylinepenalty \interdisplaylinepenalty=100
\newcount\interfootnotelinepenalty \interfootnotelinepenalty=100
\def\normalbaselines{\lineskip\normallineskip
  \baselineskip\normalbaselineskip \lineskiplimit\normallineskiplimit}
```

Macros for text

Here we introduce macros that are used for basic formatting unrelated to mathematics.

⟨ ***Provide macros for text formatting*** ⟩≡

⟨ ***Supply basic macros for text formatting*** ⟩
⟨ ***Define space macros*** ⟩
⟨ ***Supply various paragraph shapes*** ⟩
⟨ ***Define sectioning macros*** ⟩
⟨ ***Supply ragged setting*** ⟩
⟨ ***Supply 'boxing' macros*** ⟩
⟨ ***Supply strut*** ⟩
⟨ ***Provide alignment macros*** ⟩
⟨ ***Establish spacing after punctuation characters*** ⟩
⟨ ***Supply various ways to fill space*** ⟩
⟨ ***Define*** `\showhyphens` ***macro*** ⟩
⟨ ***Completing the job*** ⟩

\⟨tab⟩ and \⟨return⟩ are defined so that they expand to \⟨space⟩; this helps to prevent confusion, since all three cases look identical when displayed on most computer terminals.

⟨ ***Supply basic macros for text formatting*** ⟩≡

```
\def\^^M{\ } % control <return> = control <space>
\def\^^I{\ } % same for <tab>
```

The control sequences `\endgraf` and `\endline` are made equivalent to TEX's primitive `\par` and `\cr` operations, since it is often useful to redefine the meanings of `\par` and `\cr` themselves. Then come the definitions of `\space` (a blank space), `\empty` (a list of no tokens), and `\null` (an empty hbox).

`\bgroup` and `\egroup` are made to provide "implicit" grouping characters that turn out to be especially useful in macro definitions.

⟨ ***Supply basic macros for text formatting*** ⟩+≡

```
\let\endgraf=\par \let\endline=\cr

\def\space{ }
```

```
401     \def\empty{}
402     \def\null{\hbox{}}
403
404     \let\bgroup={ \let\egroup=}
```

The `\obeylines` macro says '`\let^^M=\par`' instead of '`\def^^M{\par}`' because the `\let` technique allows constructions such as '`\let\par=\cr \obeylines \halign{...}`' in which `\cr`'s need not be given within the alignment.

⟨ ***Supply basic macros for text formatting*** ⟩+≡

```
405     {\catcode`\^^M=\active % these lines must end with %
406       \gdef\obeylines{\catcode`\^^M\active \let^^M\par}%
407       \global\let^^M\par} % this is in case ^^M appears in a \write
408     \def\obeyspaces{\catcode`\ \active}
409     {\obeyspaces\global\let =\space}
```

The macros `\lq`, `\rq`, `\lbrack`, and `\rbrack` are defined, for people who have difficulty typing quotation marks and/or brackets.

⟨ ***Supply basic macros for text formatting*** ⟩+≡

```
410     \def\lq{`} \def\rq{'}
411     \def\lbrack{[} \def\rbrack{]}
```

The macros `\enskip`, `\quad`, and `\qquad` provide spaces that are legitimate breakpoints within a paragraph; `\enspace`, `\thinspace`, and `\negthinspace` produce space that cannot cause a break (although the space will disappear if it occurs just next to certain kinds of breaks). All six of these spaces are relative to the current font.

You can get horizontal space that never disappears by saying '`\hglue`⟨glue⟩'; this space is able to stretch or shrink. Similarly, there's a vertical analog, '`\vglue`⟨glue⟩'.

The `\nointerlineskip` macro suppresses interline glue that would ordinarily be inserted before the next box in vertical mode; this is a "one shot" macro, but `\offinterlineskip` is more drastic—it sets things up so that future interline glue will be present, but zero. There also are macros for potentially breakable vertical spaces: `\smallskip`, `\medskip`, and `\bigskip`.

⟨ ***Define space macros*** ⟩≡

```
412     \def\thinspace{\kern .16667em }
413     \def\negthinspace{\kern-.16667em }
414     \def\enspace{\kern.5em }
415
416     \def\enskip{\hskip.5em\relax}
417     \def\quad{\hskip1em\relax}
418     \def\qquad{\hskip2em\relax}
419
420     \def\smallskip{\vskip\smallskipamount}
421     \def\medskip{\vskip\medskipamount}
422     \def\bigskip{\vskip\bigskipamount}
423
424     \def\nointerlineskip{\prevdepth-1000\p@}
425     \def\offinterlineskip{\baselineskip-1000\p@
426       \lineskip\z@ \lineskiplimit\maxdimen}
427
428     \def\topglue{\nointerlineskip\vglue-\topskip\vglue} % for top of page
429     \def\vglue{\afterassignment\vgl@\skip@=}
```

```
\def\vgl@{\par \dimen@\prevdepth \hrule height\z@
  \nobreak\vskip\skip@ \prevdepth\dimen@}
\def\hglue{\afterassignment\hgl@\skip@=}
\def\hgl@{\leavevmode \count@\spacefactor \vrule width\z@
  \nobreak\hskip\skip@ \spacefactor\count@}
```

The following macros introduce penalty markers that make breaking less, or more, desirable. The `\break`, `\nobreak`, and `\allowbreak` macros are intended for use in any mode; the `~` (tie) and `\slash` (hyphen-like '/') macros are intended for horizontal mode. The others are intended only for vertical mode, i.e., between paragraphs, so they begin with `\par`.

⟨ ***Define space macros*** ⟩+≡

```
\def~{\penalty\@M \ } % tie
\def\slash{/\penalty\exhyphenpenalty} % a '/' that acts like a '-'

\def\break{\penalty-\@M}
\def\nobreak{\penalty \@M}
\def\allowbreak{\penalty \z@}

\def\filbreak{\par\vfil\penalty-200\vfilneg}
\def\goodbreak{\par\penalty-500 }
\def\eject{\par\break}
\def\supereject{\par\penalty-\@MM}

\def\removelastskip{\ifdim\lastskip=\z@\else\vskip-\lastskip\fi}
\def\smallbreak{\par\ifdim\lastskip<\smallskipamount
  \removelastskip\penalty-50\smallskip\fi}
\def\medbreak{\par\ifdim\lastskip<\medskipamount
  \removelastskip\penalty-100\medskip\fi}
\def\bigbreak{\par\ifdim\lastskip<\bigskipamount
  \removelastskip\penalty-200\bigskip\fi}
```

`\line`, `\leftline`, `\rightline`, and `\centerline` produce boxes of the full line width, while `\llap` and `\rlap` make boxes whose effective width is zero. The `\underbar` macro puts its argument into an hbox with a straight line at a fixed distance under it.

`\underbar` uses math mode to do its job, although the operation is essentially non-mathematical in nature. A few of the other macros below use math mode in similar ways; thus, TEX's mathematical abilities prove to be useful even when no mathematical typesetting is actually being done. A special control sequence `\m@th` is used to "turn off" `\mathsurround` when such constructions are being performed.)

⟨ ***Supply 'boxing' macros*** ⟩≡

```
\def\line{\hbox to\hsize}
\def\leftline#1{\line{#1\hss}}
\def\rightline#1{\line{\hss#1}}
\def\centerline#1{\line{\hss#1\hss}}

\def\rlap#1{\hbox to\z@{#1\hss}}
\def\llap#1{\hbox to\z@{\hss#1}}

\def\m@th{\mathsurround\z@}
\def\underbar#1{$\setbox\z@\hbox{#1}\dp\z@\z@
  \m@th \underline{\box\z@}$}
```

A \strut is implemented here as a rule of width zero. The '\relax' in this macro and in others below is necessary in case \strut appears first in an alignment entry, because TEX is in a somewhat unpredictable mode at such times.

⟨ *Supply strut* ⟩≡

```
\newbox\strutbox
\setbox\strutbox=\hbox{\vrule height8.5pt depth3.5pt width\z@}
\def\strut{\relax\ifmmode\copy\strutbox\else\unhcopy\strutbox\fi}
```

The \ialign macro provides for alignments when it is necessary to be sure that \tabskip is initially zero. The \hidewidth macro can be used essentially as \hfill in alignment entries that are permitted to "stick out" of their column. There's also \multispan, which permits alignment entries to span one or more columns.

⟨ *Provide alignment macros* ⟩≡

```
\def\hidewidth{\hskip\hideskip} % for alignment entries that can stick out
\def\ialign{\everycr{}\tabskip\z@skip\halign} % initialized \halign
\newcount\mscount
\def\multispan#1{\omit \mscount#1\relax
  \loop\ifnum\mscount>\@ne \sp@n\repeat}
\def\sp@n{\span\omit\advance\mscount\m@ne}
```

Now we get to the "tabbing" macros. They keep track of the tab positions by maintaining boxes full of empty boxes having the specified widths.

The macro \+ has been declared '\outer' here, so that TEX will be better able to detect runaway arguments and definitions. A non-\outer version, called \tabalign, has also been provided in case it is necessary to use \+ in some "inner" place. You can use \tabalign just like \+, except after \settabs.

⟨ *Provide alignment macros* ⟩+≡

```
\newif\ifus@ \newif\if@cr
\newbox\tabs \newbox\tabsyet \newbox\tabsdone

\def\cleartabs{\global\setbox\tabsyet\null \setbox\tabs\null}
\def\settabs{\setbox\tabs\null \futurelet\next\sett@b}
\let\+=\relax % in case this file is being read in twice
\def\sett@b{\ifx\next\+\def\nxt{\afterassignment\s@tt@b\let\nxt}%
  \else\let\nxt\s@tcols\fi \let\next\relax \nxt}
\def\s@tt@b{\let\nxt\relax \us@false\m@ketabbox}
\def\tabalign{\us@true\m@ketabbox} % non-\outer version of \+
\outer\def\+{\tabalign}
\def\s@tcols#1\columns{\count@#1\dimen@\hsize
  \loop\ifnum\count@>\z@ \@nother \repeat}
\def\@nother{\dimen@ii\dimen@ \divide\dimen@ii\count@
  \setbox\tabs\hbox{\hbox to\dimen@ii{}\unhbox\tabs}%
  \advance\dimen@-\dimen@ii \advance\count@\m@ne}

\def\m@ketabbox{\begingroup
  \global\setbox\tabsyet\copy\tabs
  \global\setbox\tabsdone\null
  \def\cr{\@crtrue\crcr\egroup\egroup
    \ifus@\unvbox\z@\lastbox\fi\endgroup
    \setbox\tabs\hbox{\unhbox\tabsyet\unhbox\tabsdone}}%
```

```
497       \setbox\z@\vbox\bgroup\@crfalse
498         \ialign\bgroup&\t@bbox##\t@bb@x\crcr}
499
500     \def\t@bbox{\setbox\z@\hbox\bgroup}
501     \def\t@bb@x{\if@cr\egroup % now \box\z@ holds the column
502       \else\hss\egroup \global\setbox\tabsyet\hbox{\unhbox\tabsyet
503           \global\setbox\@ne\lastbox}% now \box\@ne holds its size
504       \ifvoid\@ne\global\setbox\@ne\hbox to\wd\z@{}%
505       \else\setbox\z@\hbox to\wd\@ne{\unhbox\z@}\fi
506       \global\setbox\tabsdone\hbox{\box\@ne\unhbox\tabsdone}\fi
507       \box\z@}
```

Paragraph shapes of a limited but important kind are provided by `\item`, `\itemitem`, and `\narrower`. A macro `\hang` causes hanging indentation by the normal amount of `\parindent`, after the first line; thus, the entire paragraph will be indented by the same amount (unless it began with `\noindent`). `\textindent{stuff}` is like `\indent`, but it puts the '`stuff`' into the indentation, flush right except for an en space; it also removes spaces that might follow the right brace in '`{stuff}`'.

⟨ ***Supply various paragraph shapes*** ⟩≡

```
508     \def\hang{\hangindent\parindent}
509     \def\textindent#1{\indent\llap{#1\enspace}\ignorespaces}
510     \def\item{\par\hang\textindent}
511     \def\itemitem{\par\indent \hangindent2\parindent \textindent}
512     \def\narrower{\advance\leftskip\parindent
513       \advance\rightskip\parindent}
```

The `\beginsection` macro is intended to mark the beginning of a new major subdivision in a document; to use it, you say '`\beginsection`⟨section title⟩' followed by a blank line (or `\par`). The macro first emits glue and penalties, designed to start a new page if the present page is nearly full; then it makes a `\bigskip` and puts the section title flush left on a line by itself, in boldface type. The section title is also displayed on the terminal. After a `\smallskip`, with page break prohibited, a `\noindent` command is given; this suppresses indentation in the next paragraph, i.e., in the first paragraph of the new section. (However, the next "paragraph" will be empty if vertical mode material immediately follows the `\beginsection` command.)

Special statements in a mathematical paper are often called theorems, lemmas, definitions, axioms, postulates, remarks, corollaries, algorithms, facts, conjectures, or some such things, and they generally are given special typographic treatment. The `\proclaim` macro puts the title of the proclamation in boldface, then sets the rest of the paragraph in slanted type. The paragraph is followed by something similar to `\medbreak`, except that the amount of penalty is different so that page breaks are discouraged:

⟨ ***Define sectioning macros*** ⟩≡

```
514     \outer\def\beginsection#1\par{\vskip\z@ plus.3\vsize\penalty-250
515       \vskip\z@ plus-.3\vsize\bigskip\vskip\parskip
516       \message{#1}\leftline{\bf#1}\nobreak\smallskip\noindent}
517     \outer\def\proclaim #1. #2\par{\medbreak
518       \noindent{\bf#1.\enspace}{\sl#2\par}%
519       \ifdim\lastskip<\medskipamount \removelastskip\penalty55\medskip\fi}
```

Ragged-right setting is initiated by restricting the spaces between words to have a fixed width, and by putting variable space at the right of each line. You should not call `\raggedright` until your text font has already been specified.

It is assumed that the ragged-right material will not be in a variety of different sizes. If this assumption is not valid, a different approach should be used: `\fontdimen` parameters 3 and 4 of the fonts you will be using should be set to zero, by saying, e.g., '`\fontdimen3\tenrm=0pt`'. These parameters specify the stretchability and shrinkability of interword spaces. A special macro `\ttraggedright` should be used for ragged-right setting in `typewriter type`, since the spaces between words are generally bigger in that style. (Spaces are already unstretchable and unshrinkable in font `cmtt`.)

⟨ ***Supply ragged setting*** ⟩≡

```
\def\raggedright
   {\rightskip\z@ plus2em \spaceskip.3333em \xspaceskip.5em\relax}
\def\ttraggedright
   {\tt\rightskip\z@ plus2em\relax} % for use with \tt only
```

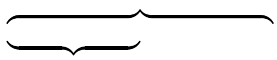
The `\nonfrenchspacing` macro is be used to change the default sfcodes (set by `INITEX`) of punctuation marks. This macro, when called, affects spacing after punctuation marks.

⟨ ***Establish spacing after punctuation characters*** ⟩≡

```
\def\frenchspacing{\sfcode`\.\@m \sfcode`\?\@m \sfcode`\!\@m
  \sfcode`\:\@m \sfcode`\;\@m \sfcode`\,\@m}
\def\nonfrenchspacing{\sfcode`\.3000\sfcode`\?3000\sfcode`\!3000%
  \sfcode`\:2000\sfcode`\;1500\sfcode`\,1250 }
```

`\rightarrowfill` and `\leftarrowfill` macros use `\cleaders` with a repeatable box consisting of the middle 10 units of a minus sign, where one unit is $\frac{1}{18}$ em. The leaders are preceded and followed by $-$ and $\rightarrow$; there's enough backspacing to compensate for up to 5 units of extra space, fore and aft, that `\cleaders` might leave blank. In this way a macro is obtained such that `\hbox to 100pt{\rightarrowfill}` yields '$\longrightarrow$'.

The `\overbrace` and `\underbrace` macros of plain TEX are constructed by combining characters with rules. Font `cmex10` contains four symbols ◟◜◝◞, each of which has depth zero and height equal to the thickness of a rule that joins them properly. Therefore it's easy to define `\upbracefill` and `\downbracefill` macros so that you can obtain, e.g.,



by saying '`\hbox to 100pt{\downbracefill}\hbox to 50pt{\upbracefill}`' in vertical mode.

⟨ ***Supply various ways to fill space*** ⟩≡

```
\def\hrulefill{\leaders\hrule\hfill}
\def\dotfill{\cleaders\hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfill}
\def\rightarrowfill{$\m@th\smash-\mkern-6mu%
  \cleaders\hbox{$\mkern-2mu\smash-\mkern-2mu$}\hfill
  \mkern-6mu\mathord\rightarrow$}
\def\leftarrowfill{$\m@th\mathord\leftarrow\mkern-6mu%
  \cleaders\hbox{$\mkern-2mu\smash-\mkern-2mu$}\hfill
  \mkern-6mu\smash-$}
\mathchardef\braceld="37A \mathchardef\bracerd="37B
\mathchardef\bracelu="37C \mathchardef\braceru="37D
\def\downbracefill{$\m@th \setbox\z@\hbox{$\braceld$}%
  \braceld\leaders\vrule height\ht\z@ depth\z@\hfill\braceru
```

```
540       \bracelu\leaders\vrule height\ht\z@ depth\z@\hfill\bracerd$}
541     \def\upbracefill{$\m@th \setbox\z@\hbox{$\braceld$}%
542       \bracelu\leaders\vrule height\ht\z@ depth\z@\hfill\bracerd
543       \braceld\leaders\vrule height\ht\z@ depth\z@\hfill\braceru$}
```

If you want to see all of the hyphens that plain TEX will find in some random text, you can say '`\showhyphens{`⟨random text⟩`}`' and the results will appear on your terminal (and in the log file). The `\showhyphens` macro creates an hbox that is intentionally underfull.

⟨ ***Define*** `\showhyphens` ***macro*** ⟩≡

```
544     \def\showhyphens#1{\setbox0\vbox{\parfillskip\z@skip\hsize\maxdimen\tenrm
545       \pretolerance\m@ne\tolerance\m@ne\hbadness0\showboxdepth0\ #1}}
```

At the end of a TEX manuscript it's usually best to finish everything off by typing '`\bye`'. The '`\vfill\supereject`' gets TEX to flush out all remaining insertions, with blank space filling the bottom of incomplete pages, and '`\end`' sends the computer into its endgame routine.

⟨ ***Completing the job*** ⟩≡

```
546     \outer\def\bye{\par\vfill\supereject\end}
```

Macros for math

Most of this section consists of long listings of special symbols together with their font locations.

⟨ ***Provide macros for math formatting*** ⟩≡

⟨ ***Define math space macros*** ⟩
⟨ ***Define Greek letters*** ⟩
⟨ ***Define math symbols*** ⟩
⟨ ***Encode large operators*** ⟩
⟨ ***Encode binary operations*** ⟩
⟨ ***Encode relations*** ⟩
⟨ ***Supply vertical and diagonal dots*** ⟩
⟨ ***Supply variable-width math accents*** ⟩
⟨ ***Supply extensible delimiters*** ⟩
⟨ ***Provide access to delimiters of various sizes*** ⟩
⟨ ***Supply common math functions*** ⟩
⟨ ***Provide*** `\cases` ***and*** `\matrix` ***macros*** ⟩
⟨ ***Provide support to typeset displayed equations*** ⟩

TEX does automatic spacing of math formulas so that they look right, and this is almost true. But occasionally you must give TEX some help. The number of possible math formulas is vast, and TEX's spacing rules are rather simple, so it is natural that exceptions should arise. The basic elements of space that TEX puts into formulas are called: thin spaces, medium spaces, thick spaces. The normal space between words of a paragraph is approximately equal to two thin spaces.

You can add your own spacing whenever you want to, by using the control sequences

`\,` thin space (normally 1/6 of a quad);
`\>` medium space (normally 2/9 of a quad);
`\;` thick space (normally 5/18 of a quad);
`\!` negative thin space (normally −1/6 of a quad).

⟨ ***Define math space macros*** ⟩≡

```
\def\,{\mskip\thinmuskip}
\def\>{\mskip\medmuskip}
\def\;{\mskip\thickmuskip}
\def\!{\mskip-\thinmuskip}
```

The next job is to define Greek letters and other symbols of type Ord. Uppercase Greek letters are assigned hexadecimal codes of the form ″7xxx, so that they will change families when `\fam` changes.

`\mathchardef` defines a control sequence to be a synonym for a math character (check `\mathchar` for the meaning of hex number).

⟨ ***Define Greek letters*** ⟩≡

```
\mathchardef\alpha="010B
\mathchardef\beta="010C
\mathchardef\gamma="010D
\mathchardef\delta="010E
\mathchardef\epsilon="010F
\mathchardef\zeta="0110
\mathchardef\eta="0111
\mathchardef\theta="0112
\mathchardef\iota="0113
\mathchardef\kappa="0114
\mathchardef\lambda="0115
\mathchardef\mu="0116
\mathchardef\nu="0117
\mathchardef\xi="0118
\mathchardef\pi="0119
\mathchardef\rho="011A
\mathchardef\sigma="011B
\mathchardef\tau="011C
\mathchardef\upsilon="011D
\mathchardef\phi="011E
\mathchardef\chi="011F
\mathchardef\psi="0120
\mathchardef\omega="0121
\mathchardef\varepsilon="0122
\mathchardef\vartheta="0123
\mathchardef\varpi="0124
\mathchardef\varrho="0125
\mathchardef\varsigma="0126
\mathchardef\varphi="0127
\mathchardef\Gamma="7000
\mathchardef\Delta="7001
\mathchardef\Theta="7002
\mathchardef\Lambda="7003
\mathchardef\Xi="7004
\mathchardef\Pi="7005
\mathchardef\Sigma="7006
\mathchardef\Upsilon="7007
\mathchardef\Phi="7008
\mathchardef\Psi="7009
\mathchardef\Omega="700A
```

⟨ *Define math symbols* ⟩≡

```
\mathchardef\aleph="0240
\def\hbar{{\mathchar'26\mkern-9muh}}
\mathchardef\imath="017B
\mathchardef\jmath="017C
\mathchardef\ell="0160
\mathchardef\wp="017D
\mathchardef\Re="023C
\mathchardef\Im="023D
\mathchardef\partial="0140
\mathchardef\infty="0231
\mathchardef\prime="0230
\mathchardef\emptyset="023B
\mathchardef\nabla="0272
\def\surd{{\mathchar"1270}}
\mathchardef\top="023E
\mathchardef\bot="023F
\def\angle{{\vbox{\ialign{$\m@th\scriptstyle##$\crcr
      \not\mathrel{\mkern14mu}\crcr
      \noalign{\nointerlineskip}
      \mkern2.5mu\leaders\hrule height.34pt\hfill\mkern2.5mu\crcr}}}}
\mathchardef\triangle="0234
\mathchardef\forall="0238
\mathchardef\exists="0239
\mathchardef\neg="023A \let\lnot=\neg
\mathchardef\flat="015B
\mathchardef\natural="015C
\mathchardef\sharp="015D
\mathchardef\clubsuit="027C
\mathchardef\diamondsuit="027D
\mathchardef\heartsuit="027E
\mathchardef\spadesuit="027F
```

Integral signs get special treatment so that their limits won't be set above and below.

⟨ *Encode large operators* ⟩≡

```
\mathchardef\coprod="1360
\mathchardef\bigvee="1357
\mathchardef\bigwedge="1356
\mathchardef\biguplus="1355
\mathchardef\bigcap="1354
\mathchardef\bigcup="1353
\mathchardef\intop="1352 \def\int{\intop\nolimits}
\mathchardef\prod="1351
\mathchardef\sum="1350
\mathchardef\bigotimes="134E
\mathchardef\bigoplus="134C
\mathchardef\bigodot="134A
\mathchardef\ointop="1348 \def\oint{\ointop\nolimits}
\mathchardef\bigsqcup="1346
\mathchardef\smallint="1273
```

⟨ ***Encode binary operations*** ⟩≡

```
\mathchardef\triangleleft="212F
\mathchardef\triangleright="212E
\mathchardef\bigtriangleup="2234
\mathchardef\bigtriangledown="2235
\mathchardef\wedge="225E \let\land=\wedge
\mathchardef\vee="225F \let\lor=\vee
\mathchardef\cap="225C
\mathchardef\cup="225B
\mathchardef\ddagger="227A
\mathchardef\dagger="2279
\mathchardef\sqcap="2275
\mathchardef\sqcup="2274
\mathchardef\uplus="225D
\mathchardef\amalg="2271
\mathchardef\diamond="2205
\mathchardef\bullet="220F
\mathchardef\wr="226F
\mathchardef\div="2204
\mathchardef\odot="220C
\mathchardef\oslash="220B
\mathchardef\otimes="220A
\mathchardef\ominus="2209
\mathchardef\oplus="2208
\mathchardef\mp="2207
\mathchardef\pm="2206
\mathchardef\circ="220E
\mathchardef\bigcirc="220D
\mathchardef\setminus="226E % for set difference A\setminus B
\mathchardef\cdot="2201
\mathchardef\ast="2203
\mathchardef\times="2202
\mathchardef\star="213F
```

Relations are also fairly straightforward, except for the ones that are constructed from other characters. The `\mapstochar` is a character '↦' of width zero that is quite useless by itself, but it combines with right arrows to make `\mapsto` '↦' and `\longmapsto` '⟼'. Similarly, `\not` is a relation character of width zero that puts a slash over the character that follows. When two relations are adjacent in a math formula, TEX puts no space between them.

⟨ ***Encode relations*** ⟩≡

```
\mathchardef\propto="322F
\mathchardef\sqsubseteq="3276
\mathchardef\sqsupseteq="3277
\mathchardef\parallel="326B
\mathchardef\mid="326A
\mathchardef\dashv="3261
\mathchardef\vdash="3260
\mathchardef\nearrow="3225
\mathchardef\searrow="3226
\mathchardef\nwarrow="322D
\mathchardef\swarrow="322E
\mathchardef\Leftrightarrow="322C
```

```
\mathchardef\Leftarrow="3228
\mathchardef\Rightarrow="3229
\def\neq{\not=} \let\ne=\neq
\mathchardef\leq="3214 \let\le=\leq
\mathchardef\geq="3215 \let\ge=\geq
\mathchardef\succ="321F
\mathchardef\prec="321E
\mathchardef\approx="3219
\mathchardef\succeq="3217
\mathchardef\preceq="3216
\mathchardef\supset="321B
\mathchardef\subset="321A
\mathchardef\supseteq="3213
\mathchardef\subseteq="3212
\mathchardef\in="3232
\mathchardef\ni="3233 \let\owns=\ni
\mathchardef\gg="321D
\mathchardef\ll="321C
\mathchardef\not="3236
\mathchardef\leftrightarrow="3224
\mathchardef\leftarrow="3220 \let\gets=\leftarrow
\mathchardef\rightarrow="3221 \let\to=\rightarrow
\mathchardef\mapstochar="3237 \def\mapsto{\mapstochar\rightarrow}
\mathchardef\sim="3218
\mathchardef\simeq="3227
\mathchardef\perp="323F
\mathchardef\equiv="3211
\mathchardef\asymp="3210
\mathchardef\smile="315E
\mathchardef\frown="315F
\mathchardef\leftharpoonup="3128
\mathchardef\leftharpoondown="3129
\mathchardef\rightharpoonup="312A
\mathchardef\rightharpoondown="312B

\def\joinrel{\mathrel{\mkern-3mu}}
\def\relbar{\mathrel{\smash-}} % \smash, because - has the same height as +
\def\Relbar{\mathrel=}
\mathchardef\lhook="312C \def\hookrightarrow{\lhook\joinrel\rightarrow}
\mathchardef\rhook="312D \def\hookleftarrow{\leftarrow\joinrel\rhook}
\def\bowtie{\mathrel\triangleright\joinrel\mathrel\triangleleft}
\def\models{\mathrel|\joinrel=}
\def\Longrightarrow{\Relbar\joinrel\Rightarrow}
\def\longrightarrow{\relbar\joinrel\rightarrow}
\def\longleftarrow{\leftarrow\joinrel\relbar}
\def\Longleftarrow{\Leftarrow\joinrel\Relbar}
\def\longmapsto{\mapstochar\longrightarrow}
\def\longleftrightarrow{\leftarrow\joinrel\rightarrow}
\def\Longleftrightarrow{\Leftarrow\joinrel\Rightarrow}
\def\iff{\;\Longleftrightarrow\;}
```

After defining characters \ldotp and \cdotp that act as math punctuation, \ldots and \cdots macros are defined that give the proper spacing in most circumstances. Vertical and diagonal dots (\vdots and \ddots) are also provided here:

⟨ *Supply vertical and diagonal dots* ⟩≡

```
\mathchardef\ldotp="613A % ldot as a punctuation mark
\mathchardef\cdotp="6201 % cdot as a punctuation mark
\mathchardef\colon="603A % colon as a punctuation mark
\def\ldots{\mathinner{\ldotp\ldotp\ldotp}}
\def\cdots{\mathinner{\cdotp\cdotp\cdotp}}
\def\vdots{\vbox{\baselineskip4\p@ \lineskiplimit\z@
    \kern6\p@\hbox{.}\hbox{.}\hbox{.}}}
\def\ddots{\mathinner{\mkern1mu\raise7\p@\vbox{\kern7\p@\hbox{.}}\mkern2mu
    \raise4\p@\hbox{.}\mkern2mu\raise\p@\hbox{.}\mkern1mu}}
```

Most of the math accents are handled entirely by the \mathaccent primitive, but a few of the variable-width ones are constructed the hard way.

⟨ *Supply variable-width math accents* ⟩≡

```
\def\acute{\mathaccent"7013 }
\def\grave{\mathaccent"7012 }
\def\ddot{\mathaccent"707F }
\def\tilde{\mathaccent"707E }
\def\bar{\mathaccent"7016 }
\def\breve{\mathaccent"7015 }
\def\check{\mathaccent"7014 }
\def\hat{\mathaccent"705E }
\def\vec{\mathaccent"017E }
\def\dot{\mathaccent"705F }
\def\widetilde{\mathaccent"0365 }
\def\widehat{\mathaccent"0362 }
\def\overrightarrow#1{\vbox{\m@th\ialign{##\crcr
      \rightarrowfill\crcr\noalign{\kern-\p@\nointerlineskip}
      $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overleftarrow#1{\vbox{\m@th\ialign{##\crcr
      \leftarrowfill\crcr\noalign{\kern-\p@\nointerlineskip}
      $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overbrace#1{\mathop{\vbox{\m@th\ialign{##\crcr\noalign{\kern3\p@}
      \downbracefill\crcr\noalign{\kern3\p@\nointerlineskip}
      $\hfil\displaystyle{#1}\hfil$\crcr}}}\limits}
\def\underbrace#1{\mathop{\vtop{\m@th\ialign{##\crcr
      $\hfil\displaystyle{#1}\hfil$\crcr\noalign{\kern3\p@\nointerlineskip}
      \upbracefill\crcr\noalign{\kern3\p@}}}}\limits}
\def\skew#1#2#3{{\muskip\z@#1mu\divide\muskip\z@\tw@ \mkern\muskip\z@
    #2{\mkern-\muskip\z@{#3}\mkern\muskip\z@}\mkern-\muskip\z@}{}}
```

Now we come to 24 delimiters that can change their size. These are denoted explicitly by a ⟨27-bit number⟩.

If we denote 27-bit number by "cqrstuv, then delimiter codes are assigned by

\delimiter "cqrstuv, where

c — the class
q — the font family number of

`rs` — the position of the the small variant of the delimiter
`t` — the font family number of
`uv` — the position of the the large variant of the delimiter

After `\left` and `\right` commands the class digit is ignored. When TEX is not looking for a delimiter the righmost three digits `tuv` are ignored, and the remaining four `cqrs` are treated as a `\mathchar`.

⟨ ***Supply extensible delimiters*** ⟩≡

```
\def\lmoustache{\delimiter"437A340 } % top from (, bottom from )
\def\rmoustache{\delimiter"537B341 } % top from ), bottom from (
\def\lgroup{\delimiter"462833A } % extensible ( with sharper tips
\def\rgroup{\delimiter"562933B } % extensible ) with sharper tips
\def\arrowvert{\delimiter"26A33C } % arrow without arrowheads
\def\Arrowvert{\delimiter"26B33D } % double arrow without arrowheads
\def\bracevert{\delimiter"77C33E } % the vertical bar that extends braces
\def\Vert{\delimiter"26B30D } \let\|=\Vert
\def\vert{\delimiter"26A30C }
\def\uparrow{\delimiter"3222378 }
\def\downarrow{\delimiter"3223379 }
\def\updownarrow{\delimiter"326C33F }
\def\Uparrow{\delimiter"322A37E }
\def\Downarrow{\delimiter"322B37F }
\def\Updownarrow{\delimiter"326D377 }
\def\backslash{\delimiter"26E30F } % for double coset G\backslash H
\def\rangle{\delimiter"526930B }
\def\langle{\delimiter"426830A }
\def\rbrace{\delimiter"5267309 } \let\}=\rbrace
\def\lbrace{\delimiter"4266308 } \let\{=\lbrace
\def\rceil{\delimiter"5265307 }
\def\lceil{\delimiter"4264306 }
\def\rfloor{\delimiter"5263305 }
\def\lfloor{\delimiter"4262304 }
```

In the `plain.tex` format and in the Computer Modern math fonts there is only one radical—the square root. The meaning of `\radical` is analogous to the `\delimiter` commands. Only the class number is dropped. Joining the radical character and the horizontal rule is done by leeting the radical character have a large depth, and the height which is equal to the rule thickness. The rule is placed on the baseline and the radical character is placed below. Then the whole is centered around math axis.

⟨ ***Supply extensible delimiters*** ⟩+≡

```
\def\choose{\atopwithdelims()}
\def\brack{\atopwithdelims[]}
\def\brace{\atopwithdelims\{\}}

\def\sqrt{\radical"270370 }
```

These macros depend on actual sizes of delimiters.

⟨ *Provide access to delimiters of various sizes* ⟩≡

```
\def\bigl{\mathopen\big}
\def\bigm{\mathrel\big}
\def\bigr{\mathclose\big}
\def\Bigl{\mathopen\Big}
\def\Bigm{\mathrel\Big}
\def\Bigr{\mathclose\Big}
\def\biggl{\mathopen\bigg}
\def\biggm{\mathrel\bigg}
\def\biggr{\mathclose\bigg}
\def\Biggl{\mathopen\Bigg}
\def\Biggm{\mathrel\Bigg}
\def\Biggr{\mathclose\Bigg}
\def\big#1{{\hbox{$\left#1\vbox to8.5\p@{}\right.\n@space$}}}
\def\Big#1{{\hbox{$\left#1\vbox to11.5\p@{}\right.\n@space$}}}
\def\bigg#1{{\hbox{$\left#1\vbox to14.5\p@{}\right.\n@space$}}}
\def\Bigg#1{{\hbox{$\left#1\vbox to17.5\p@{}\right.\n@space$}}}
\def\n@space{\nulldelimiterspace\z@ \m@th}
```

The `\mathpalette` operation constructs a formula in all four styles; it is applied here in the implementation of `\phantom`, `\smash`, `\root`, and other operations. (Actually `\phantom` and `\smash` are not perfect: They assume that the current style is uncramped.)

These definitions illustrate how other built-up symbol combinations could be defined to work in all four styles.

The control sequences `\sp` and `\sb` are provided for people who can't easily type `^` and `_`; a "discretionary times sign" `\*` is defined.

⟨ *Provide macros for math formatting* ⟩+≡

```
\def\mathpalette#1#2{\mathchoice{#1\displaystyle{#2}}%
  {#1\textstyle{#2}}{#1\scriptstyle{#2}}{#1\scriptscriptstyle{#2}}}
\newbox\rootbox
\def\root#1\of{\setbox\rootbox\hbox{$\m@th\scriptscriptstyle{#1}$}
  \mathpalette\r@@t}
\def\r@@t#1#2{\setbox\z@\hbox{$\m@th#1\sqrt{#2}$}
  \dimen@\ht\z@ \advance\dimen@-\dp\z@
  \mkern5mu\raise.6\dimen@\copy\rootbox \mkern-10mu \box\z@}
\newif\ifv@ \newif\ifh@
\def\vphantom{\v@true\h@false\ph@nt}
\def\hphantom{\v@false\h@true\ph@nt}
\def\phantom{\v@true\h@true\ph@nt}
\def\ph@nt{\ifmmode\def\next{\mathpalette\mathph@nt}%
  \else\let\next\makeph@nt\fi\next}
\def\makeph@nt#1{\setbox\z@\hbox{#1}\finph@nt}
\def\mathph@nt#1#2{\setbox\z@\hbox{$\m@th#1{#2}$}\finph@nt}
\def\finph@nt{\setbox\tw@\null
  \ifv@ \ht\tw@\ht\z@ \dp\tw@\dp\z@\fi
  \ifh@ \wd\tw@\wd\z@\fi \box\tw@}
\def\mathstrut{\vphantom(}
\def\smash{\relax % \relax, in case this comes first in \halign
  \ifmmode\def\next{\mathpalette\mathsm@sh}\else\let\next\makesm@sh
  \fi\next}
\def\makesm@sh#1{\setbox\z@\hbox{#1}\finsm@sh}
\def\mathsm@sh#1#2{\setbox\z@\hbox{$\m@th#1{#2}$}\finsm@sh}
```

```
\def\finsm@sh{\ht\z@\z@ \dp\z@\z@ \box\z@}

\def\cong{\mathrel{\mathpalette\@vereq\sim}} % congruence sign
\def\@vereq#1#2{\lower.5\p@\vbox{\lineskiplimit\maxdimen\lineskip-.5\p@
    \ialign{$\m@th#1\hfil##\hfil$\crcr#2\crcr=\crcr}}}
\def\notin{\mathrel{\mathpalette\c@ncel\in}}
\def\c@ncel#1#2{\m@th\ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
\def\rightleftharpoons{\mathrel{\mathpalette\rlh@{}}}
\def\rlh@#1{\vcenter{\m@th\hbox{\ooalign{\raise2pt
          \hbox{$#1\rightharpoonup$}\crcr
        $#1\leftharpoondown$}}}}
\def\buildrel#1\over#2{\mathrel{\mathop{\kern\z@#2}\limits^{#1}}}
\def\doteq{\buildrel\textstyle.\over=}

\let\sp=^ \let\sb=_
\def\*{\discretionary{\thinspace\the\textfont2\char2}{}{}}
```

The 32 common functions whose names generally appear in roman letters.

⟨ ***Supply common math functions*** ⟩≡

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

```

```
\def\bmod{\nonscript\mskip-\medmuskip\mkern5mu
  \mathbin{\rm mod}\penalty900\mkern5mu\nonscript\mskip-\medmuskip}
\def\pmod#1{\allowbreak\mkern18mu({\rm mod}\,\,#1)}
```

The definition of `\matrix` goes to some pains to ensure that two n-rowed matrices will have the same height and the same depth, unless at least one of their rows is unusually big. The definition of `\bordermatrix` is even more complicated, but it seems to work reasonably well; it uses a constant `\p@renwd` that represents the width of a big extensible left parenthesis.

⟨ ***Provide*** `\cases` ***and*** `\matrix` ***macros*** ⟩≡

```
\def\cases#1{\left\{\,\vcenter{\normalbaselines\m@th
    \ialign{$##\hfil$&\quad##\hfil\crcr#1\crcr}}\right.}
\def\matrix#1{\null\,\vcenter{\normalbaselines\m@th
    \ialign{\hfil$##$\hfil&&\quad\hfil$##$\hfil\crcr
      \mathstrut\crcr\noalign{\kern-\baselineskip}
      #1\crcr\mathstrut\crcr\noalign{\kern-\baselineskip}}}\,}
\def\pmatrix#1{\left(\matrix{#1}\right)}
\newdimen\p@renwd
\setbox0=\hbox{\tenex B} \p@renwd=\wd0 % width of the big left (
\def\bordermatrix#1{\begingroup \m@th
  \setbox\z@\vbox{\def\cr{\crcr\noalign{\kern2\p@\global\let\cr\endline}}%
    \ialign{$##$\hfil\kern2\p@\kern\p@renwd&\thinspace\hfil$##$\hfil
      &&\quad\hfil$##$\hfil\crcr
      \omit\strut\hfil\crcr\noalign{\kern-\baselineskip}%
      #1\crcr\omit\strut\cr}}%
  \setbox\tw@\vbox{\unvcopy\z@\global\setbox\@ne\lastbox}%
  \setbox\tw@\hbox{\unhbox\@ne\unskip\global\setbox\@ne\lastbox}%
  \setbox\tw@\hbox{$\kern\wd\@ne\kern-\p@renwd\left(\kern-\wd\@ne
    \global\setbox\@ne\vbox{\box\@ne\kern2\p@}%
    \vcenter{\kern-\ht\@ne\unvbox\z@\kern-\baselineskip}\,\right)$}%
  \null\;\vbox{\kern\ht\@ne\box\tw@}\endgroup}
```

The value of `\lineskiplimit` is assumed to be `\normallineskiplimit` plus the accumulated amount of "opening up." Thus, the `\vskip` instructions in `\displ@y` will compensate for the fact that the first baseline of an alignment is separated by an opened-up baselineskip from the last line preceding the display.

⟨ ***Provide support to typeset displayed equations*** ⟩≡

```
\def\openup{\afterassignment\@penup\dimen@=}
\def\@penup{\advance\lineskip\dimen@
  \advance\baselineskip\dimen@
  \advance\lineskiplimit\dimen@}
\def\eqalign#1{\null\,\vcenter{\openup\jot\m@th
  \ialign{\strut\hfil$\displaystyle{##}$&$\displaystyle{{}##}$\hfil
      \crcr#1\crcr}}\,}
\newif\ifdt@p
\def\displ@y{\global\dt@ptrue\openup\jot\m@th
  \everycr{\noalign{\ifdt@p \global\dt@pfalse \ifdim\prevdepth>-1000\p@
      \vskip-\lineskiplimit \vskip\normallineskiplimit \fi
      \else \penalty\interdisplaylinepenalty \fi}}
\def\@lign{\tabskip\z@skip\everycr{}} % restore inside \displ@y
\def\displaylines#1{\displ@y \tabskip\z@skip
  \halign{\hbox to\displaywidth{$\@lign\hfil\displaystyle##\hfil$}\crcr
```

```
      #1\crcr}}
  \def\eqalignno#1{\displ@y \tabskip\centering
    \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip\z@skip
      &$\@lign\displaystyle{{}##}$\hfil\tabskip\centering
      &\llap{$\@lign##$}\tabskip\z@skip\crcr
      #1\crcr}}
  \def\leqalignno#1{\displ@y \tabskip\centering
    \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip\z@skip
      &$\@lign\displaystyle{{}##}$\hfil\tabskip\centering
      &\kern-\displaywidth\rlap{$\@lign##$}\tabskip\displaywidth\crcr
      #1\crcr}}
```

Below we have an interesting set of macros that convert `f'''` into `f^{\prime\prime\prime}`.

⟨ ***Supply common math functions*** ⟩+≡

```
{\catcode`\'=\active \gdef'{^\bgroup\prim@s}}
\def\prim@s{\prime\futurelet\next\pr@m@s}
\def\pr@m@s{\ifx'\next\let\nxt\pr@@@s \else\ifx^\next\let\nxt\pr@@@t
  \else\let\nxt\egroup\fi\fi \nxt}
\def\pr@@@s#1{\prim@s} \def\pr@@@t#1#2{#2\egroup}
{\catcode`\^^Z=\active \gdef^^Z{\not=}} % ^^Z is like \ne in math
{\catcode`\_=\active \global\let_=\_} % _ in math is either subscript or \_
```

Macros for output

⟨ ***Prepare page for output*** ⟩≡

⟨ ***Supply headers and footers*** ⟩
⟨ ***Supply footnotes*** ⟩
⟨ ***Supply floating insertions*** ⟩
⟨ ***Supply raggedbottom setting*** ⟩
⟨ ***Set up the output routine*** ⟩

The `\makeheadline` macro constructs a vbox of height and depth zero. The magic constant −22.5 pt is equal to

$$\texttt{\topskip} - \langle\text{height of strut}\rangle - 2\,\texttt{\baselineskip}$$

i.e., 10 pt − 8.5 pt − 24 pt (assuming default values of `\topskip` and the height of the strut); this places the reference point of the headline exactly 24 pt above the reference point of the top line on the page, unless the headline or the top line are excessively large.

The `\advancepageno` macro normally advances `\pageno` by +1; but if `\pageno` is negative (for roman numerals), the advance is by −1. The new value of `\pageno` will be appropriate for the next time the output routine is called into action.

⟨ ***Supply headers and footers*** ⟩≡

```
\countdef\pageno=0 \pageno=1 % first page is number 1
\newtoks\headline \headline={\hfil} % headline is normally blank
\newtoks\footline \footline={\hss\tenrm\folio\hss}
% footline is normally a centered page number in font \tenrm
%
\def\makeheadline{\vbox to\z@{\vskip-22.5\p@
  \line{\vbox to8.5\p@{}\the\headline}\vss}\nointerlineskip}
```

```
\def\makefootline{\baselineskip24\p@\line{\the\footline}}
%
\def\folio{\ifnum\pageno<\z@ \romannumeral-\pageno \else\number\pageno \fi}
\def\nopagenumbers{\footline{\hfil}} % blank out the footline
\def\advancepageno{\ifnum\pageno<\z@ \global\advance\pageno\m@ne
  \else\global\advance\pageno\@ne \fi} % increase |pageno|
```

Ragged-bottom setting is achieved by inserting infinite glue, which overpowers the stretchability of `\topskip`. This macros assume that `\topskip = 10pt`

⟨ ***Supply raggedbottom setting*** ⟩≡

```
\newif\ifr@ggedbottom
\def\raggedbottom{\topskip 10\p@ plus60\p@ \r@ggedbottomtrue}
\def\normalbottom{\topskip 10\p@ \r@ggedbottomfalse} % undoes \raggedbottom
```

There are 255 classes of insertions, `\insert0` to `\insert254`, and they are tied to other registers of the same number. For example, `\insert100` is connected with `\count100`, `\dimen100`, `\skip100`, and `\box100`.

For our purposes let's consider a particular class of insertions called class n; we will then be dealing with TEX's primitive command

`\insert`n{⟨vertical mode material⟩}

which puts an insertion item into a horizontal or vertical list. For this class of insertions

`\box`n	is where the material appears when a page is output;
`\count`n	is the magnification factor for page breaking;
`\dimen`n	is the maximum insertion size per page;
`\skip`n	is the extra space to allocate on a page.

For example, material inserted with `\insert100` will eventually appear in `\box100`.

`\footnote` macro depends on the value of `\bigskipamount`, and `\parindent`. Because the value of `\vsize` (8.9 in) is greater than `\dimen\footins` (8 in), footnotes never fill up the whole page.

⟨ ***Supply footnotes*** ⟩≡

```
\newinsert\footins
\def\footnote#1{\let\@sf\empty % parameter #2 (the text) is read later
  \ifhmode\edef\@sf{\spacefactor\the\spacefactor}\/\fi
  #1\@sf\vfootnote{#1}}
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty\interfootnotelinepenalty
  \splittopskip\ht\strutbox % top baseline for broken footnotes
  \splitmaxdepth\dp\strutbox \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip \spaceskip\z@skip \xspaceskip\z@skip
  \textindent{#1}\footstrut\futurelet\next\fo@t}
\def\fo@t{\ifcat\bgroup\noexpand\next \let\next\f@@t
  \else\let\next\f@t\fi \next}
\def\f@@t{\bgroup\aftergroup\@foot\let\next}
\def\f@t#1{#1\@foot}
\def\@foot{\strut\egroup}
\def\footstrut{\vbox to\splittopskip{}}

\skip\footins=\bigskipamount % space added when footnote is present
```

```
977     \count\footins=1000 % footnote magnification factor (1 to 1)
978     \dimen\footins=8in % maximum footnotes per page
979
980     \def\footnoterule{\kern-3\p@
981       \hrule width 2truein \kern 2.6\p@} % the \hrule is .4pt high
```

Here the constant `12\p@ = 12pt` is hard coded into the format.

⟨ ***Supply floating insertions*** ⟩≡

```
982     \newinsert\topins
983     \newif\ifp@ge \newif\if@mid
984     \def\topinsert{\@midfalse\p@gefalse\@ins}
985     \def\midinsert{\@midtrue\@ins}
986     \def\pageinsert{\@midfalse\p@getrue\@ins}
987     \skip\topins=\z@skip % no space added when a topinsert is present
988     \count\topins=1000 % magnification factor (1 to 1)
989     \dimen\topins=\maxdimen % no limit per page
990     \def\@ins{\par\begingroup\setbox\z@\vbox\bgroup} % start a \vbox
991     \def\endinsert{\egroup % finish the \vbox
992       \if@mid \dimen@\ht\z@ \advance\dimen@\dp\z@ \advance\dimen@12\p@
993         \advance\dimen@\pagetotal \advance\dimen@-\pageshrink
994         \ifdim\dimen@>\pagegoal\@midfalse\p@gefalse\fi\fi
995       \if@mid \bigskip\box\z@\bigbreak
996       \else\insert\topins{\penalty100 % floating insertion
997         \splittopskip\z@skip
998         \splitmaxdepth\maxdimen \floatingpenalty\z@
999         \ifp@ge \dimen@\dp\z@
1000        \vbox to\vsize{\unvbox\z@\kern-\dimen@}% depth is zero
1001        \else \box\z@\nobreak\bigskip\fi}\fi\endgroup}
```

The value of `\boxmaxdepth` is set to `\maxdepth` so that the vbox will be constructed under the assumptions that TEX's page builder has used to set up `\box255`.

The `\pagecontents` macro produces a vertical list for everything that belongs on the main body of the page, namely the contents of `\box255` together with illustrations (inserted at the top) and footnotes (inserted at the bottom). `\topins` and `\footins` are the insertion class numbers for the two kinds of insertions used in plain TEX; if more classes of insertions are added, `\pagecontents` should be changed accordingly. Notice that the boxes are unboxed so that the glue coming from insertions can help out the glue on the main page. The `\footnoterule` macro places a dividing line between the page and its footnotes; it makes a net contribution of 0 pt to the height of the vertical list.

The `\dosupereject` macro is designed to clear out any insertions that have been held over, whether they are illustrations or footnotes or both. The negative `\kern` here cancels out the natural space of the `\topskip` glue that goes above the empty `\line`; that empty line box prevents the `\vfill` from disappearing into a page break. The vertical list that results from `\dosupereject` is placed on TEX's list of things to put out next, just after the straggling insertions have been reconsidered. Hence another super-eject will occur, and the process will continue until no insertions remain.

⟨ ***Set up the output routine*** ⟩≡

```
1002    \output{\plainoutput}
1003    \def\plainoutput{\shipout\vbox{\makeheadline\pagebody\makefootline}%
1004      \advancepageno
1005      \ifnum\outputpenalty>-\@MM \else\dosupereject\fi}
1006
```

```
\def\pagebody{\vbox to\vsize{\boxmaxdepth\maxdepth \pagecontents}}

\def\dosupereject{\ifnum\insertpenalties>\z@ % something is being held over
  \line{}\kern-\topskip\nobreak\vfill\supereject\fi}

\def\pagecontents{\ifvoid\topins\else\unvbox\topins\fi
  \dimen@=\dp\@cclv \unvbox\@cclv % open up \box255
  \ifvoid\footins\else % footnote info is present
    \vskip\skip\footins
    \footnoterule
    \unvbox\footins\fi
  \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
```

Hyphenation

A discretionary break consists of three sequences of characters called the *pre-break*, *post-break*, and *no-break* texts. The idea is that if a line break occurs here, the pre-break text will appear at the end of the current line and the post-break text will occur at the beginning of the next line; but if no break occurs, the no-break text will appear in the current line. The discretionary are specified by writing

`\discretionary{`⟨pre-break text⟩`}{`⟨post-break text⟩`}{`⟨no-break text⟩`}`

where the three texts consist entirely of characters, boxes, and kerns. If a word contains discretionary breaks TEX will not hyphenate it.

Hyphenation exceptions are specified with the statements like

```
\hyphenation{gal-axy iso-peri-met-ric}
```

which gives to TEX locations where these words may be hyphenated.

The default values of `\hyphenchar` is `‘-`.

Sometimes the typewriter fonts are given `\hyphenchar`⟨font name⟩`=-1` which value inhibits hyphenation.

⟨ ***Read hyphenation patterns*** ⟩≡

```
\lefthyphenmin=2 \righthyphenmin=3 % disallow x- or -xx breaks
\input hyphen
```

Initalization

⟨ ***Initialize the layout*** ⟩≡

```
\normalbaselines\rm % select roman font
\nonfrenchspacing % punctuation affects the spacing
```

Programming support

⟨ *Provide programming constructs* ⟩≡

⟨ *Supply loops and conditionals* ⟩
⟨ *Define* `\tracingall` *macro* ⟩

The `\loop...\repeat` macro provides for iterative operations. In this macro and several others, the control sequence '`\next`' is given a temporary value that is not going to be needed later; thus, `\next` acts like a "scratch control sequence."

The macro `\newif` to be used for definitions of new conditionals. For example, `\newif\iffoo` creates `\footrue`, `\foofalse` to go with `\iffoo`.

⟨ *Supply loops and conditionals* ⟩≡

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next\iterate \else\let\next\relax\fi \next}
\let\repeat=\fi % this makes \loop...\if...\repeat skippable

\outer\def\newif#1{\count@\escapechar \escapechar\m@ne
  \expandafter\expandafter\expandafter
   \edef\@if#1{true}{\let\noexpand#1=\noexpand\iftrue}%
  \expandafter\expandafter\expandafter
   \edef\@if#1{false}{\let\noexpand#1=\noexpand\iffalse}%
  \@if#1{false}\escapechar\count@} % the condition starts out false
\def\@if#1#2{\csname\expandafter\if@\string#1#2\endcsname}
{\uccode`1=`i \uccode`2=`f \uppercase{\gdef\if@12{}}} % `if' is required
```

⟨ *Define* `\tracingall` *macro* ⟩≡

```
\def\tracingall{\tracingonline\@ne\tracingcommands\tw@\tracingstats\tw@
  \tracingpages\@ne\tracingoutput\@ne\tracinglostchars\@ne
  \tracingmacros\tw@\tracingparagraphs\@ne\tracingrestores\@ne
  \showboxbreadth\maxdimen\showboxdepth\maxdimen\errorstopmode}
```

The format name and version number are recorded in control sequences, in order to help the people who might have to explain why something doesn't work.

⟨ *Identify the format* ⟩≡

```
\def\fmtname{plain+W}\def\fmtversion{3.14159+W}
```

Appendices

Efficiency and memory-space considerations

One difficulty with large sets of macros is that they take up space. It would be nice to preload every macro that every TEX user has ever dreamed up; but there might not be enough room, because TEX's memory capacity is finite. You might find it necessary to hold back and to load only the macros that are really needed.

How much memory space does a macro require?

There are four kinds of memory involved: token memory, name memory, string memory, and character memory. (If any of these becomes too full, it will be necessary to increase what TEX calls the macro memory size, the hash size, the number of strings, and/or the pool size, respectively. The token memory is most important; a macro takes one cell of token memory for each token in its definition, including the '`{`' and the '`}`'. For example, the comparatively short definition

```
\def\example#1\two{\four}
```

takes five tokens: `#1`, `\two`, $\texttt{\{}_1$, `\four`, and $\texttt{\}}_2$. Each control sequence also takes up one cell of name memory, one cell of string memory, and as many cells of character memory as there are characters in the name (seven in the case of `\example`). Character memory is comparatively cheap; four characters, or in some cases five, will fit in the same number of bits as a single cell of token memory, inside the machine. Therefore you don't save much by choosing short macro names.

TEX will tell you how close you come to exceeding its current memory capacity if you say `\tracingstats=1`. TEX governs fourteen kinds of memory:

`number of strings` (names of control sequences and files)
`pool size` (the characters in such names)
`main memory size` (boxes, glue, breakpoints, token lists, characters, etc.)
`hash size` (control sequence names)
`font memory` (font metric data)
`exception dictionary` (hyphenation exceptions)
`input stack size` (simultaneous input sources)
`semantic nest size` (unfinished lists being constructed)
`parameter stack size` (macro parameters)
`buffer size` (characters in lines being read from files)
`save size` (values to restore at group ends)
`text input levels` (`\input` files and error insertions)
`grouping levels` (unfinished groups)
`pattern memory` (hyphenation pattern data)

The current amount of memory available will also be shown.

One obvious way to keep from loading too many macros is to keep the macro files short and to `\input` only the ones that you need.

Extensible delimiters

TEX builds large delimiters by using "extensible" characters, which are specified by giving top, midle, bottom, and repeatable characters in an **extensible** command. For example, the extensible left parentheses in `cmex10` are defined by (see Figure 4)

```
extensible oct"060": oct"060", 0, oct"100", oct"102";
```

this says that character code `oct"060"` specifies an extensible delimiter constructed from itself as the top piece, from character number `oct"100"` as the bottom piece, and from character number `oct"102"` as the piece which should be repeated as often as necessary to reach a desired size. In this particular example there is no middle piece, but characters like curly braces have a middle piece as well. A zero value in the top, middle, or bottom position means that no character should be used in that part of the construction; but a zero value in the final position means that character number zero is the repeater. The width of an extensible character is taken to be the width of the repeater.

Also several characters of various sizes can be linked together in a series by means of a **charlist** command. For example (see Figure 4),

```
charlist oct"000": oct"020": oct"022": oct"040";
```

is used in the font `cmex10` to specify the left parentheses that TEX uses in displayed math formulas. TEX follows charlist to make variable-size delimiters and variable-size accents, as well to link `\textstyle` and the `\displaystyle` operators.

Font dimensions

The main information about font consists of the dimensions of the characters. These numbers TEX finds in the font metric files. Except character dimensions, font metric files contain: values for `\fontdimen` parameters, italic correction of characters, ligature and kerning programs for characters. We change the fontdimen parameters with the (global) assignment:

`\fontdimen`⟨number⟩⟨font⟩⟨equals⟩⟨dimen⟩,

for example, the assignment `\fontdimen8\tenex = 0.6pt` changes width of the fraction bar from default 0.4 pt to 0.6 pt.

The first seven fontdimen parameters have the following meaning:

1. the slant per point
2. the interword space; that is used unless `\spaceskip` is specified
3. interword stretch
4. interword shrink
5. the x-height
6. the quad width (for the font in family 2, 1/18 th quad width is equal to 1 mu)
7. the extra space; that value is added to the interword space used whenever `\spacefactor` $\geq$ 2000, unless `\xspaceskip` is specified.

For the font in family 2 attributes 8–19 specify positioning of fractions, subscripts, superscripts.

fraction numerator attributes: minimum shift up, from the main baseline, of the baseline of the numerator of a generalized fraction

8. num1: for display style
9. num2: for text style or smaller if a fraction bar is present
10. num3: for text style or smaller if no a fraction bar is present

fraction denominator attributes: minimum shift down, from the main baseline, of the baseline of the denominator of a generalized fraction

11. denom1: for display style
12. denom2: for text style or smaller

superscript attributes: minimum shift up, from the main baseline, of the baseline of the superscript

13. sup1: for display style
14. sup2: for text style or smaller, non-cramped
15. sup3: for text style or smaller, cramped

subscript attributes: minimum shift down, from the main baseline, of the baseline of a subscript

16. sub1: when no superscript is present
17. sub2: when superscript is present

script adjustment attributes: for use only with non-glyph, that is, composite objects

18. sup_drop: maximum distance of superscript baseline below top of nucleus
19. sub_drop: minimum distance of subscript baseline below bottom of nucleus

Delimiter span attributes: height plus depth of delimiter enclosing a generalized fraction.

20. delim1: in display style
21. delim2: in text style or smaller

The last parameter, the height of the math axis, specifies the height above the baseline of the fraction bar, and the centre of large delimiters and most operators and relations. This position is used in vertical centering.

22. axis_height.

For the font in family 3 attributes 9–12 determine extra space added when limits are attached to operators. The attribute 8 specifies thickness of the rule used for overlines, underlines, radical extenders, and fraction bars. From that dimension are derived 'clearances' around fraction bar. The attribute 13 specifies extra space added above and below attached limits.

8. default_rule_thickness
9. big_op_spacing1
10. big_op_spacing2
11. big_op_spacing3
12. big_op_spacing4
13. big_op_spacing5

We have: big_op_spacing1(2) $\leq$ space between upper (lower) limit and top (bottom) of large operator $\leq$ big_op_spacing3(4)

fontdimen	cmsy10	cmex10	cmtt10	cmr10	cmti10	cmbx10
1.	0.25pt	0.0pt	0.0pt	0.0pt	0.25pt	0.0pt
2.	0.0pt	0.0pt	5.24995pt	3.33333pt	3.57774pt	3.83331pt
3.	0.0pt	0.0pt	0.0pt	1.66666pt	1.53333pt	1.91666pt
4.	0.0pt	0.0pt	0.0pt	1.11111pt	1.0222pt	1.27777pt
5.	4.30554pt	4.30554pt	4.30554pt	4.30554pt	4.30554pt	4.44444pt
6.	10.00002pt	10.00002pt	10.4999pt	10.00002pt	10.22217pt	11.49994pt
7.	0.0pt	0.0pt	5.24995pt	1.11111pt	1.0222pt	1.27777pt
8.	6.76508pt	0.39998pt				
9.	3.93732pt	1.11111pt				
10.	4.4373pt	1.66666pt				
11.	6.85951pt	1.99998pt				
12.	3.44841pt	6.0pt				

```
13.    4.12892pt  1.0pt

14.    3.62892pt
15.    2.88889pt
16.    1.49998pt
17.    2.47217pt
18.    3.86108pt
19.    0.5pt
20.   23.9pt
21.   10.09999pt
22.    2.5pt
```

Font tables

Figure 1. `cmr10`—family 0

	´0	*´1*	*´2*	*´3*	*´4*	*´5*	*´6*	*´7*	
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
´01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	
´02x	ı	ȷ	`	´	ˇ	˘	¯	˚	″1x
´03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	˝	!	”	#	$	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	¡	=	¿	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	“	]	ˆ	˙	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	–	—	″	˜	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

Figure 2. `cmmi10`—family 1

	′0	′1	′2	′3	′4	′5	′6	′7	
′00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
′01x	Φ	Ψ	Ω	α	β	γ	δ	ϵ	
′02x	ζ	η	θ	ι	κ	λ	μ	ν	″1x
′03x	ξ	π	ρ	σ	τ	υ	ϕ	χ	
′04x	ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ	″2x
′05x	$\leftharpoonup$	$\leftharpoondown$	$\rightharpoonup$	$\rightharpoondown$	‘	’	$\triangleright$	$\triangleleft$	
′06x	0	1	2	3	4	5	6	7	″3x
′07x	8	9	.	,	<	/	>	$\star$	
′10x	∂	A	B	C	D	E	F	G	″4x
′11x	H	I	J	K	L	M	N	O	
′12x	P	Q	R	S	T	U	V	W	″5x
′13x	X	Y	Z	$\flat$	$\natural$	$\sharp$	$\smile$	$\frown$	
′14x	ℓ	a	b	c	d	e	f	g	″6x
′15x	h	i	j	k	l	m	n	o	
′16x	p	q	r	s	t	u	v	w	″7x
′17x	x	y	z	$\imath$	$\jmath$	$\wp$	$\vec{}$	$\frown$	
	″8	″9	″A	″B	″C	″D	″E	″F	

Figure 3. `cmsy10`—family 2

	′0	′1	′2	′3	′4	′5	′6	′7	
′00x	−	·	×	∗	÷	◇	±	∓	″0x
′01x	⊕	⊖	⊗	⊘	⊙	◯	∘	•	
′02x	≍	≡	⊆	⊇	≤	≥	≼	≽	″1x
′03x	∼	≈	⊂	⊃	≪	≫	≺	≻	
′04x	←	→	↑	↓	↔	↗	↘	≃	″2x
′05x	⇐	⇒	⇑	⇓	⇔	↖	↙	∝	
′06x	′	∞	∈	∋	△	▽	/	♭	″3x
′07x	∀	∃	¬	∅	ℜ	ℑ	⊤	⊥	
′10x	ℵ	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$	″4x
′11x	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$	
′12x	$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	″5x
′13x	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	∪	∩	⊎	∧	∨	
′14x	⊢	⊣	⌊	⌋	⌈	⌉	{	}	″6x
′15x	⟨	⟩	|	‖	↕	⇕	\	≀	
′16x	√	⨿	∇	∫	⊔	⊓	⊑	⊒	″7x
′17x	§	†	‡	¶	♣	♢	♡	♠	
	″8	″9	″A	″B	″C	″D	″E	″F	

Figure 4. cmex10—family 3

	′0	′1	′2	′3	′4	′5	′6	′7	
′00x	(	)	[	]	⌊	⌋	⌈	⌉	″0x
′01x	{	}	⟨	⟩	∣	∥	/	\	
′02x	(	)	(	)	[	]	⌊	⌋	″1x
′03x	⌈	⌉	{	}	⟨	⟩	/	\	
′04x	(	)	[	]	⌊	⌋	⌈	⌉	″2x
′05x	{	}	⟨	⟩	/	\	/	\	
′06x	⎛	⎞	⎡	⎤	⎣	⎦	∣	∣	″3x
′07x	⎧	⎫	⎩	⎭	⎨	⎬	∣	∣	
′10x	⎝	⎠	∣	∣	⟨	⟩	⨆	⨆	″4x
′11x	∮	∮	⊙	⊙	⊕	⊕	⊗	⊗	
′12x	∑	∏	∫	⋃	⋂	⊎	⋀	⋁	″5x
′13x	∑	∏	∫	⋃	⋂	⊎	⋀	⋁	
′14x	∐	∐	^	⏜	⏜	~	~	⏜	″6x
′15x	[	]	⌊	⌋	⌈	⌉	{	}	
′16x	√	√	√	√	√	∣	⌜	∥	″7x
′17x	↑	↓	⏜	⏝	⏝	⏜	⇑	⇓	
	″8	″9	″A	″B	″C	″D	″E	″F	

User-level definitions

Definitions

Chunks