Typesetting Mathematics — User's Guide†

Brian W. Kernighan
Lorinda L. Cherry

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This is the user's guide for a system for typesetting mathematics. Mathematical expressions are described in a language designed to be easy to use by people who know neither mathematics nor typesetting. Enough of the language to set in-line expressions like $\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$ or display equations like

$$G(z) = e^{\ln G(z)} = \exp \left[\sum_{k \geq 1} \frac{S_k z^k}{k} \right] = \prod_{k \geq 1} e^{S_k z^k / k}$$

$$= \left[1 + S_1 z + \frac{S_1^2 z^2}{2!} + \ldots \right] \left[1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \ldots \right] \cdots$$

$$= \sum_{m \geq 0} \left[\sum_{\substack{k_1, k_2, \ldots, k_m \geq 0 \\ k_1 + 2k_2 + \ldots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \, \frac{S_2^{k_2}}{2^{k_2} k_2!} \cdots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right] z^m$$

can be learned in an hour or so.

The language interfaces directly with the formatting language so mathematical expressions can be embedded in the running text of a manuscript, and the entire document produced in one process. This user's guide is an example of its output.

1. Introduction

is a program and a language for typesetting mathematics. The language was designed to be easy to use by people who know neither mathematics nor typesetting. Thus knows relatively little about mathematics. In particular, mathematical symbols like +, −, ×, parentheses, and so on have no special meanings. is quite happy to set garbage (but it will look good).

is a preprocessor for the typesetter formatter so the normal mode of operation is to prepare a document with both mathematics and ordinary text interspersed, and let set the mathematics while does the body of the text.

† This is a revised version of |reference(eqn cstr).

To use on UNIX,®

```
eqn files | troff
```

The design and implementation of *eqn* is discussed in |reference(kernighan cherry cacm).

2. Display Equations

To tell where a mathematical expression begins and ends, one marks it with lines beginning `.EQ` and `.EN`. Thus if you type the lines

```
.EQ
x=y+z
.EN
```

your output will look like

$$x = y + z$$

The `.EQ` and `.EN` are copied through untouched; they are not otherwise processed by This means that you have to take care of things like centering, numbering, and so on yourself. The most common way is to use a macro package like `-ms` or `-mm`, which allow you to center, indent, left-justify, and number equations.

With the `-ms` package, equations are centered by default. To left-justify an equation, use `.EQ L` instead of `.EQ`. To indent it, use `.EQ I`. Any of these can be followed by an arbitrary 'equation number' which will be placed at the right margin. For example, the input

```
.EQ I (3.1a)
x = f(y/2) + y/2
.EN
```

produces the output

$$x=f(y/2)+y/2 \tag{3.1a}$$

For stupid historical reasons, the `-mm` macros package requires you to surround `.EQ` and `.EN` with a pair of totally unnecessary commands, `.DS` and `.DE`:

```
.DS        for -mm only
.EQ
your equation
.EN
.DE
```

There is also a shorthand notation so in-line expressions like π_i^2 can be entered without `.EQ` and `.EN`. We will talk about it in section 9.

3. Input spaces

Spaces, tabs, and newlines within an expression are thrown away by (Normal text is left absolutely alone.) Thus between `.EQ` and `.EN`,

```
x=y+z
```

and

```
x = y + z
```

and

```
x    =    y
   + z
```

and so on all produce the same output:

$$x=y+z$$

You should use spaces and newlines freely to make your input equations readable and easy to edit. In particular, very long lines are a bad idea, since they are often hard to fix if you make a mistake.

4. Output spaces

To force extra spaces into the use a tilde ''~'' for each space you want:

```
x~=~y~+~z
```

gives

$$x~=~y~+~z$$

You can also use a circumflex ''^'', which gives a space half the width of a tilde. It is mainly useful for fine-tuning.

5. Symbols, Special Names, Greek

knows some mathematical symbols, some mathematical names, and the Greek alphabet. For example,

```
x=2 pi int sin ( omega t)dt
```

produces

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

Here the spaces in the input are necessary to tell that `int`, `pi`, `sin`, and `omega` are separate entities that should get special treatment. The `sin`, digit 2, and parentheses are set in roman type instead of italic; `pi` and `omega` are made Greek; and `int` becomes the integral sign.

When in doubt, leave spaces around separate parts of the input. A common error is to type `f(pi)` without leaving spaces on both sides of the `pi`. As a result, does not recognize `pi` as a special word, and you get $f(pi)$ instead of $f(\pi)$.

A list of names appears in section 24. Knowledgeable users can also use four-character names for anything doesn't know about, like `\(L1` for the AT&T death-star.

6. Spaces, Again

The only way can deduce that some sequence of letters might be special is if that sequence is separated from the letters on either side of it. This can be done by surrounding a special word by spaces, tabs, or newlines, as we did in the previous section.

You can also make special words stand out by surrounding them with tildes or circumflexes:

```
x~=~2~pi~int~sin~(~omega~t~)~dt
```

is much the same as the last example, except that the tildes not only separate the magic words like `sin`, `omega`, and so on, but also add extra spaces, one space per tilde:

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

Special words can also be separated by braces `{}` and double quotes `""`, which have special meanings that we will see soon.

7. Subscripts and Superscripts

Subscripts and superscripts are obtained with the words `sub` and `sup`.

```
x sup 2 + y sub k
```

gives

$$x^2 + y_k$$

takes care of all the size changes and vertical motions needed to make the output look right. The words `sub` and `sup` must be surrounded by spaces; `x sub2` will give you *xsub*2 instead of x_2. Furthermore, don't forget to leave a space (or a tilde, etc.) to mark the end of a subscript or superscript. A common error is to say something like

```
y = (x sup 2)+1
```

which causes

$$y = (x^{2)+1}$$

instead of the intended

$$y = (x^2) + 1$$

Subscripted subscripts and superscripted superscripts also work:

```
x sub i sub 1
```

is

$$x_{i_1}$$

A subscript and superscript on the same thing are printed one above the other if the subscript comes

```
x sub i sup 2
```

is

$$x_i^2$$

Other than this special case, `sub` and `sup` group to the right, so means x^{y_z}, not $x^y{}_z$.

8. Braces for Grouping

Normally, the end of a subscript or superscript is marked simply by a blank (or tab or tilde, etc.) What if the subscript or superscript is something that has to be typed with blanks in it? In that case, you can use the braces { and } to mark the beginning and end of the subscript or superscript:

```
e sup {i omega t}
```

is

$$e^{i \omega t}$$

Rule: Braces can always be used to force to treat something as a unit, or just to make your intent perfectly clear. Thus:

```
x sub {i sub 1} sup 2
```

is

$$x_{i_1}^2$$

with braces, but

```
x sub i sub 1 sup 2
```

is

$$x_{i_1^2}$$

which is rather different.

Braces can occur within braces:

```
e sup {i pi sup {rho +1}}
```

is

$$e^{i \pi^{\rho + 1}}$$

The general rule is that anywhere you could use some single thing like `x`, you can use an arbitrarily complicated thing if you enclose it in braces. will look after all the details of positioning it and making it the right size.

In all cases, make sure you have the right number of braces. Leaving one out or adding an extra will cause to complain bitterly.

Occasionally you will have to print braces. To do this, enclose them in double quotes, like `"{"`. Quoting is discussed in more detail in section 15.

9. Shorthand for In-line Equations

With the `-ms` and `-mm` macro packages, `.EQ` and `.EN` imply a displayed equation. But in most documents, it is necessary to follow mathematical conventions not just in display equations, but also in the body of the text, for example by making variable names like *x* italic.

provides a shorthand for short in-line expressions. You can define two characters to mark the left and right ends of an in-line equation, and then type expressions right in the middle of text lines. To set both the left and right characters to dollar signs, for example, add to the beginning of your document the three lines

```
.EQ
delim $$
.EN
```

Having done this, you can then say things like

```
Let $alpha sub i$ be positive, and
let $beta$ be zero.  Then we can
show that $alpha sub 1$ is $>=0$.
```

This works as you might expect—spaces, newlines, and so on are significant in the text, but not in the equation part itself. Multiple equations can occur in a single input line.

Enough room is left before and after a line that contains in-line expressions that something tall like $\sum\limits_{i=1}^{n} x_i$ does not interfere with the lines surrounding it.

To turn off the delimiters,

```
.EQ
delim off
.EN
```

Warning: don't use braces, tildes, circumflexes, or double quotes as delimiters—chaos will result.

10. Fractions

To make a fraction, use the word `over`:

```
a+b over 2c = 1
```

gives

$$\frac{a+b}{2c} = 1$$

The line is made the right length and positioned automatically. Braces can be used to make clear what goes over what:

```
{alpha + beta} over {sin (x)}
```

is

$$\frac{\alpha+\beta}{\sin(x)}$$

What happens when there is both an `over` and a `sup` in the same expression? In such an apparently ambiguous case, does the `sup` before the `over`, so

```
-b sup 2 over pi
```

is $\frac{-b^2}{\pi}$ instead of $-b^{\frac{2}{\pi}}$. The rules that decide which operation is done first in cases like this are summarized in section 24. When in doubt, however, use braces to make clear what goes with what.

11. Square Roots

To draw a square root, use `sqrt`:

```
sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}
```

is

$$\sqrt{a+b} + \frac{1}{\sqrt{ax^2+bx+c}}$$

Warning—square roots of tall quantities look bad, because a root-sign big enough to cover the quantity is too dark and heavy:

```
sqrt {a sup 2 over b sub 2}
```

is

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

Big square roots are generally better written as something to the power ½:

$$(a^2/b_2)^{½}$$

which is

```
(a sup 2 /b sub 2 ) sup half
```

12. Summation, Integral, Etc.

Summations, integrals, and similar constructions are easy:

```
sum from i=0 to {i= inf} x sup i
```

produces

$$\sum_{i=0}^{i=\infty} x^i$$

Notice that we used braces to indicate where the upper part $i=\infty$ begins and ends. No braces were necessary for the lower part $i=0$, because it contained no blanks. The braces will never hurt, and if the `from` and `to` parts contain any blanks, you must use braces around them.

The `from` and `to` parts are both optional, but if both are used, they have to occur in that order.

Other useful characters can replace the `sum` in our example:

```
int   prod   union   inter
```

become, respectively,

$$\int \qquad \prod \qquad \bigcup \qquad \bigcap$$

Since the thing before the `from` can be anything, even something in braces, `from-to` can often be used in unexpected ways:

```
lim from {n -> inf} x sub n =0
```

is

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

Notice the difference between `from-to` and `sub-sup`:

```
int from a to b ~~~ int sub a sup b
```

is

$$\int\limits_a^b \quad \int_a^b$$

13. Size and Font Changes

By default, equations are set in the current point size (this text is 10-point type), with standard mathematical conventions to determine what characters are in roman and what in italic. Although makes a valiant attempt to use aesthetically pleasing sizes and fonts, it is not perfect. To change sizes and fonts, use `size` and `roman`, `italic`, `bold` and `fat`. Like `sub` and `sup`, size and font changes affect only the thing that follows them, and revert to the normal situation at the end of it. Thus

```
bold x y
```

is

$$\mathbf{x}y$$

and

```
size 12 bold x = y +
   size 12 {alpha + beta}
```

gives

$$\mathbf{x} = y + \alpha + \beta$$

As always, you can use braces if you want to affect something more complicated than a single letter. For example, you can change the size of an entire equation by

```
size 8 { ... }
```

You can also change the size by a given amount; for example, you can say `size +2` to make the size two points bigger, or `size -3` to make it three points smaller. This has the advantage that you don't have to know what the current size is.

If you are using fonts other than roman, italic and bold, you can say `font` where is a name or number for the font, one or two characters long. Since is tuned for roman, italic and bold, other fonts may not give quite as good an appearance.

The `fat` operation takes the current font and widens it by overstriking: `fat grad` is ∇ and `fat {x sub i}` is $\boldsymbol{x_i}$.

If an entire document is to be in a non-standard size or font, it is a severe nuisance to have to write out a size and font change for each equation. Accordingly, you can set a ''global'' size or font which thereafter affects all equations. At the beginning of any equation, you might say, for instance,

```
.EQ
gsize 16
gfont R
...
.EN
```

to set the size to 16 and the font to roman thereafter. In place of R, you can use any of the font names. The size after `gsize` can be a relative change with `+` or `-`.

Generally, `gsize` and `gfont` will appear at the beginning of a document but they can also appear throughout a document: the global font and size can be changed as often as needed.‡

Since manages most size changes automatically, `gsize` is rarely necessary.

14. Diacritical Marks

To get diacritical marks on top of letters, there are several words:

`x dot`	$\dot{x}$
`x dotdot`	$\ddot{x}$
`x hat`	$\hat{x}$
`x tilde`	$\tilde{x}$
`x vec`	$\vec{x}$
`x dyad`	$\overleftrightarrow{x}$
`x bar`	$\bar{x}$
`x under`	$\underline{x}$
`x utilde`	$\underset{\sim}{x}$

The diacritical mark is placed at the right height. The `bar` and `under` are made the right length for the entire construct, as in $\overline{x+y+z}$; other marks are centered.

Sometimes guesses wrong on the height for bars. As an interim fix, the words `highbar` and `lowbar` are synonymous with `bar,` but force the bar to be either high (as with $\overline{X}$) or low (as with $\bar{x}$) regardless of the apparent height of the object. This is useful for constructions like $\overline{x_1}$, which is `{x sub 1} lowbar`.

15. Quoted Text

Any input entirely within quotes `"..."` is not subject to any of the normal font changes and spacing adjustments. This provides a way to do your own spacing and adjusting if needed:

‡In a footnote like this one, in which we have a few random expressions like x_i and π^2, handles the size change automatically.

```
    italic "sin(x)" + sin (x)
```

is

$$\mathit{sin(x)} + \sin(x)$$

Quotes are also used to get braces and other keywords printed:

```
    "{ size alpha }"
```

is

{ size alpha }

and

```
    roman "{ size alpha }"
```

is

{ size alpha }

The construction `""` is often used as a placeholder when grammatically needs something, but you don't actually want anything in your output. For example, to make ^{2}He, you can't just type `sup 2 roman He` because a `sup` has to be a superscript something. Thus you must say

```
    "" sup 2 roman He
```

To get a literal quote use `"\""`. If delimiters are set with `delim`, you can include them in a quoted string too. special-character names like `\(bs` can appear unquoted, but all other constructions should always be quoted.

16. Lining up Equations

Sometimes it's necessary to line up a series of equations at some horizontal position, often at an equals sign. This is done with two operations called `mark` and `lineup`.

The word `mark` may appear once at any place in an equation. It remembers the horizontal position where it appeared. Successive equations can contain one occurrence of the word `lineup`. The place where `lineup` appears is made to line up with the place marked by the previous `mark` if possible. Thus, for example, you can say

```
    .EQ I
    x+y mark = z
    .EN
    .EQ I
    x lineup = 1
    .EN
```

to produce

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

For reasons too complicated to talk about, when you use and `-ms`, use either `.EQ I` or `.EQ L`; `mark` and `lineup` don't work with centered equations. Also bear in mind that `mark` doesn't look ahead;

```
    x mark =1
    ...
    x+y lineup =z
```

will not work if there isn't room for the `x+y` part after the `mark` remembers where the `x` is. If you use `.EQ I`, that may suffice; if not, use tildes to move the first equation to the right.

17. Big Brackets, Etc.

To get big brackets, braces, parentheses, and bars around things, use the `left` and `right` commands:

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

is

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

The resulting brackets are made big enough to cover whatever they enclose. Other characters can be used besides these, but are not likely to look very good. The `floor` and `ceiling` characters are exceptions:

```
  left floor x over y right floor <=
  left ceiling a over b right ceiling
```

produces

$$\left\lfloor \frac{x}{y} \right\rfloor \leqq \left\lceil \frac{a}{b} \right\rceil$$

Several warnings about brackets are in order. First, braces are typically bigger than brackets and parentheses, because they are made up of three, five, seven, etc., pieces, while brackets can be made up of two, three, etc. Second, big left and right parentheses often look poor, because the character set is poorly designed.

The `right` part may be omitted: a ‘‘left something’’ need not have a corresponding ‘‘right something’’. If the `right` part is omitted, put braces around the thing you want the left bracket to encompass. Otherwise, the resulting brackets may be too large.

If you want to omit the `left` part, things are more complicated, because technically you can't have a `right` without a corresponding `left`. Instead you have to say

```
    left "" ... right )
```

for example. The `left ""` means a ‘‘left nothing’’. This satisfies the rules without hurting your output.

18. Piles

There is a general facility for making vertical piles of things; it comes in several flavors. For example:

```
    A ~=~ left [
      pile { a above b above c }
      ~~ pile { x above y above z }
    right ]
```

will make

$$A ~=~ \left[\begin{matrix} a & x \\ b & y \\ c & z \end{matrix} \right]$$

The elements of the pile (there can be as many as you want) are centered one above another, at the right height for most purposes. The keyword `above` is used to separate the pieces; braces are used around the entire list. The elements of a pile can be as complicated as needed, even containing more piles.

Three other forms of pile exist: `lpile` makes a pile with the elements left-justified; `rpile` makes a right-justified pile; and `cpile` makes a centered pile, just like `pile.` The vertical spacing between the pieces is somewhat larger for `l`, `r`, and `cpiles` than it is for ordinary piles.

```
  roman sign (x)~=~
  left {
     lpile {1 above 0 above -1}
     ~~ lpile
      {if~x>0 above if~x=0 above if~x<0}
```

makes

$$\mathrm{sign}(x) ~=~ \left\{ \begin{array}{ll} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{array} \right.$$

Notice the left brace without a matching right one.

It is possible to change the default vertical separation between elements of a pile, by adding a number immediately after the word `lpile`, etc. The number is in units of 1/100’s of an em (an em is about the width of the letter ‘m’); `lpile 33` makes the spacing 1/3 of an em.

19. Matrices

It is also possible to make matrices. For example, to make a neat array like

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

you have to type

```
    matrix {
      ccol { x sub i above y sub i }
      ccol { x sup 2 above y sup 2 }
    }
```

This produces a matrix with two centered columns. The elements of the columns are then listed just as for a pile, each element separated by the word `above.` You can also use `lcol` or `rcol` to left or right adjust columns. Each column can be separately adjusted, and there can be as many columns as you like.

The words `pile` and `col`, `lpile` and `lcol`, etc., are synonyms.

The reason for using a matrix instead of two adjacent piles, by the way, is that if the elements of the piles don’t all have the same height, they won’t line up properly. A matrix forces them to line up, because it looks at the entire structure before deciding what spacing to use.

A word of warning about matrices—each column must have the same number of elements in it. The world will end if you get this wrong.

20. Local Motions

Although tries to print things at the right place, it isn’t perfect, and occasionally you will need to tune the output to make it just right. Small extra horizontal spaces can be obtained with tilde and circumflex. You can also say `back` and `fwd` to move small amounts horizontally. is how far to move in 1/100’s of an em. Thus `back 50` moves back about half the width of an m. For example, `<<` produces $<<$, while `< back 50 <` produces $\ll$.

Similarly you can move things up or down with `up` and `down`

```
    x sub down 20 i sup up 20 j
```

produces x_i^j instead of the usual x_i^j.

As with `sub` or `sup`, the local motions affect the next thing in the input, and this can be something arbitrarily complicated if it is enclosed in braces.

21. Definitions

provides a facility so you can give a frequently-used string of characters a name, and thereafter just type the name instead of the whole string. For example, if the sequence

```
    x sub i + y sub i
```

appears repeatedly throughout a paper, you can save re-typing it each time by defining it like this:

```
define xy ’x sub i + y sub i’
```

This makes `xy` a shorthand for whatever characters occur between the single quotes in the definition. You can use any character instead of quote to mark the ends of the definition, so long as it doesn't appear inside the definition.

Now you can use `xy` like this:

```
.EQ
f(x) = xy ...
.EN
```

and so on. Each occurrence of `xy` will expand into what it was defined as. Be careful to leave spaces or their equivalent around the name when you actually use it, so will be able to identify it as special.

There are several things to watch out for. First, although definitions can use previous definitions, as in

```
.EQ
define xi ′ x sub i ′
define xi1 ′ xi sub 1 ′
.EN
```

Don't define something in terms of itself! A favorite error is to say

```
define X ′ roman X ′
```

This is a guaranteed disaster, since `X` is now defined in terms of itself. If you say

```
define X ′ roman "X" ′
```

however, the quotes protect the second `X`, and everything works fine.

keywords can be redefined. You can make `/` mean `over` by saying

```
define / ′ over ′
```

or redefine `over` as `/` with

```
define over ′ / ′
```

It is possible to create a parameterized definition, which will expand into something different each time it is used. If a definition contains occurrences of `$1`, `$2`, etc., as in

```
define f ′font $1 {"$2"}′
```

then subsequent uses of that name that are followed by a parenthesized list of values will have the values substituted for the `$`*n*'s:

```
f(CW, a phrase)
```

will print `a phrase` in the constant-width font `CW`.

You can test whether something is defined with `ifdef`:

`ifdef f /` *anything at all* `/`

is replaced by if `f` is defined.

22. File Inclusion

You can include a file of input as part of an equation by the construction

`include "`*filename*`"`

For historical reasons, `copy` is a synonym for `include`.

23. A Large Example

Here is the complete source for the three display equations in the abstract of this guide.

```
.EQ I
G(z)~mark =~ e sup { ln ~ G(z) }
~=~ exp left (
sum from k>=1 {S sub k z sup k} over k right )
~=~  prod from k>=1 e sup {S sub k z sup k /k}
.EN
.EQ I
lineup = left ( 1 + S sub 1 z +
{ S sub 1 sup 2 z sup 2 } over 2! + ... right )
left ( 1+ { S sub 2 z sup 2 } over 2
+ { S sub 2 sup 2 z sup 4 } over { 2 sup 2 cdot 2! }
+ ... right ) ...
.EN
.EQ I
lineup =  sum from m>=0 left (
sum from
pile { k sub 1 ,k sub 2 ,..., k sub m  >=0
above
k sub 1 +2k sub 2 + ... +mk sub m =m}
{ S sub 1 sup {k sub 1} } over {1 sup k sub 1 k sub 1 ! } ~
{ S sub 2 sup {k sub 2} } over {2 sup k sub 2 k sub 2 ! } ~
...
{ S sub m sup {k sub m} } over {m sup k sub m k sub m ! }
right ) z sup m
.EN
```

24. Keywords, Precedences, Etc.

Here are the keywords of in order of decreasing precedence:

```
dyad vec under utilde bar tilde
          hat dot dotdot
left  right
fwd   back   down  up
fat   roman  italic  bold  size
sub   sup
sqrt  over
from  to
```

These operations group to the left:

```
over  sqrt  left  right
```

All others group to the right.

Digits, parentheses, brackets, punctuation marks, and these mathematical words are converted to Roman font when encountered:

```
sin  cos  tan  sinh  cosh  tanh  arc
max  min  lim  log   ln    exp
Re   Im   and  if    for   det
```

These character sequences are recognized and translated as shown.

`>=`	≧	`nothing`	
`<=`	≦	`cdot`	·
`==`	≡	`times`	×
`!=`	≠	`del`	∇
`+-`	±	`grad`	∇
`->`	→	`...`	…
`<-`	←	`,...,`	,…,
`<<`	< <	`sum`	∑
`>>`	> >	`int`	∫
`inf`	∞	`prod`	∏
`partial`	∂	`union`	⋃
`half`	½	`inter`	⋂
`prime`	′	`dollar`	$
`approx`	≈		

To obtain Greek letters, simply spell them out in whatever case you want:

`DELTA`	Δ	`iota`	ι
`GAMMA`	Γ	`kappa`	κ
`LAMBDA`	Λ	`lambda`	λ
`OMEGA`	Ω	`mu`	μ
`PHI`	Φ	`nu`	ν
`PI`	Π	`omega`	ω
`PSI`	Ψ	`omicron`	ο
`SIGMA`	Σ	`phi`	ϕ
`THETA`	Θ	`pi`	π
`UPSILON`	ϒ	`psi`	ψ
`XI`	Ξ	`rho`	ρ
`alpha`	α	`sigma`	σ
`beta`	β	`tau`	τ
`chi`	χ	`theta`	θ
`delta`	δ	`upsilon`	υ
`epsilon`	ϵ	`xi`	ξ
`eta`	η	`zeta`	ζ
`gamma`	γ		

really ought to know the whole Greek alphabet, but the missing upper-case characters are identical to Roman ones.

These are all the words known to except for characters with names.

`above`	`fat`	`lcol`	`size`
`back`	`font`	`left`	`space`
`bar`	`from`	`lineup`	`sqrt`
`bold`	`fwd`	`lowbar`	`sub`
`ccol`	`gfont`	`lpile`	`sum`
`col`	`gsize`	`mark`	`sup`
`copy`	`hat`	`matrix`	`tilde`
`cpile`	`highbar`	`over`	`to`
`define`	`ifdef`	`pile`	`under`
`delim`	`include`	`prod`	`union`
`dot`	`int`	`rcol`	`up`
`dotdot`	`integral`	`right`	`utilde`
`down`	`inter`	`roman`	`vec`
`dyad`	`italic`	`rpile`	

25. Troubleshooting

If you make a mistake in an equation, like leaving out a brace (very common) or having one too many (very common) or having a `sup` with nothing before it (common), will tell you with the message

```
syntax error near line n, file f,
  context is >>> something <<<
```

where is approximately the line where the trouble occurred, is the name of the input file, and is a guess about what input was in error. The line number and context is approximate—look nearby as well. There are also self-explanatory messages that arise if you leave out a quote or try to run on a non-existent file.

If you want to check a document before actually printing it,

```
eqn files >/dev/null
```

will throw away the output but print the messages.

If you use something like dollar signs as delimiters, it is easy to leave one out. This causes very strange troubles. The program `checkeq` checks for misplaced or missing delimiters and similar troubles.

In-line equations can only be so big because of an internal limit in If you get the message ‘‘word overflow,’’ you have exceeded this limit. If you print the equation as a displayed equation this message will usually go away. The message ‘‘line overflow’’ indicates you have exceeded a different, bigger internal limit. The only cure for this is to break the equation into two separate ones.

On a related topic, does not break equations by itself—you must split long equations up across multiple lines by yourself, marking each by a separate `.EQ` ... `.EN` sequence.

26. Use on UNIX

To print a document that contains mathematics,

```
eqn files | troff
```

If there are any options, they go after the part of the command. For example,

```
eqn files | troff -ms
```

assumes the output is aimed at a Postscript device. If you are using some other typesetter or printer, you have to tell with the same `-T` argument that you use with

```
eqn -Taps files | troff -Taps ...
```

also uses the environment variable `TYPESETTER`; the default is `post`, as it is for can be used with the program for setting tables that contain mathematics. Use before like this:

```
    tbl files | eqn | troff
```

When is used with or the additional vertical space that adds to a line to keep it from colliding with adjacent lines will conflict with and spacing. The phrase `space 0` at the beginning of an equation will turn off the extra space.

Acknowledgements

We are deeply indebted to the late J. F. Ossanna, the author of for his willingness to extend to make our task easier, and for his continuous assistance during the development and evolution of We are also grateful to A. V. Aho for advice on language design, to S. C. Johnson for assistance with the compiler-compiler, and to all the users who have made helpful suggestions and criticisms.

References

|reference_placement

Appendix 1. Tuning *eqn* Output

There are numerous values that can be adjusted to tune the output of for a particular output device. In general, it is necessary to have access to the source code to do this tuning.

Some values can be set without access to the source. The names shown below (together with their default values) control the positioning of diacritical marks and special characters. They can be defined with any string value, as in

```
define vec_def " \v’-.5m’\s-3\(->\s0\v’.5m’
vec_def         \v’-.5m’\s-3\(->\s0\v’.5m’
dyad_def        \v’-.5m’\s-3\z\(<-\|\(->\s0\v’.5m’
hat_def         \v’-.05m’\s+1^\s0\v’.05m’
tilde_def       \v’-.05m’\s+1~\s0\v’.05m’
dot_def         \v’-.67m’.\v’.67m’
dotdot_def      \v’-.67m’..\v’.67m’
utilde_def      \v’1.0m’\s+2~\s-2\v’-1.0m’
sum_def         \|\v’.3m’\s+5\(*S\s-5\v’-.3m’\|
union_def       \|\v’.3m’\s+5\(cu\s-5\v’-.3m’\|
inter_def       \|\v’.3m’\s+5\(ca\s-5\v’-.3m’\|
prod_def        \|\v’.3m’\s+5\(*P\s-5\v’-.3m’\|
int_def         \v’.1m’\s+4\(is\s-4\v’-.1m’
Subbase         0.2
Supbase         0.4
```

The amount by which a subscript is shifted down and a superscript shifted up are controlled by `Subbase` and `Supbase`.