O rozboru jednoho makra

Denis Roegel

O rozboru jednoho makra (tutoriál)

Denis Roegel

Abstrakt

V článku dopodrobna rozebíráme makro naprogramované v TEXu, které vypočítá a vypíše prvních n prvočísel. To nám dává prostor k poukázání na některé technické detaily, které jsou častokrát začátečníky přehlíženy. Redakce navíc přidala shrnutí příkazů Plain TEXu jako přílohu tohoto článku.

Klíčová slova: Plain TEX, prvočísla.

doi: 10.5300/2010-1-2/68

This article is dedicated to Chrystel Barraband
for whom the first version was written in 1993.

Introduction

A TEX macro can be seen as the definition of a command by other commands. Both the definition of a command and the way arguments are passed obey rules which are both precise and simple, but which are often overlooked, though indispensable to a good understanding of TEX.

Moreover, the call of a TEX macro is a very different process from what happens in classical languages. It is similar to a macro call in the C preprocessor and it is hard to imagine programming with such a language! A macro call merely entails a replacement or a substitution, but it can also call other macros, including itself, which allows recursion.

Computing prime numbers

We will focus on the computation of prime numbers. $n > 1$ is prime if n is divisible only by itself and 1. If n is odd, it is sufficient to divide n by $3, 5, 7, \ldots, p \leq \lfloor\sqrt{n}\rfloor$. For, if n can be divided by $p > \lfloor\sqrt{n}\rfloor$, then n can also be divided by $q < \lfloor\sqrt{n}\rfloor$. The divisors p will be tried until $p^2 > n$.

Macros

The following example, from *The TEXbook* [1], is of an advanced level but will allow us to go straight to the heart of the matter. The macro `\primes` makes it

*Previously published in *TUGboat* **22**:1/2 (March/June 2001), pp. 78–82; translation by the author from the original in *Les Cahiers GUTenberg* number 31, December 1998, pp. 19–27.

possible to determine the first n prime numbers, starting with 2. For instance, `\primes{30}` returns the first 30 prime numbers. Here are all the definitions.

We will then analyze them in detail:

```
\newif\ifprime \newif\ifunknown
\newcount\n \newcount\p
\newcount\d \newcount\a
\def\primes#1{2,~3% assume that #1>2
  \n=#1 \advance\n by-2 % n more to go
  \p=5 % odd primes starting with p
  \loop\ifnum\n>0 \printifprime
       \advance\p by2 \repeat}
\def\printp{, % invoked if p is prime
  \ifnum\n=1 and~\fi
  \number\p \advance\n by -1 }
\def\printifprime{\testprimality
                  \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
  \loop\trialdivision
     \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{\a=\p \divide\a by\d
  \ifnum\a>\d \unknowntrue
  \else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse
              \unknownfalse\fi}
```

Declarations

First, we declare two booleans, or more precisely two tests.

`\newif\ifprime`

`\ifprime` is equivalent to `\iftrue` if "`prime`" is true. This boolean will make it possible to see if a number must be printed; thus, in `\printifprime`, the expression `\ifprime\printp\fi` means that if `\ifprime` is evaluated to `\iftrue`, then `\printp` (that is, the macro that will print the number of interest to us, namely `\p`) will be executed, otherwise nothing will happen.

`\newif\ifunknown`

"`unknown`" will be true if we are not yet sure whether `\p` is composed or not. Neither is known. Initially, "`unknown`" is thus true and the `\ifunknown` test succeeds. If "`unknown`" is false, we have knowledge about `\p`'s primality, that is, we know if `\p` is prime or not.

Next, we define a few integer variables useful in what follows:

- `\newcount\n`
  `\n` is the number of prime numbers that remain to be printed.
- `\newcount\p`
  `\p` is the current number for which primality is tested.
- `\newcount\d`
  `\d` is a variable containing the sequence of trials of divisors of `\p`.
- `\newcount\a`
  `\a` is an auxiliary variable.

Main macro

The main macro is `\primes`. It takes an argument. When the macro is defined, this argument has the name `#1`. If there were a second argument, it would be `#2`, etc. (It is not possible to have — directly — more than nine arguments; indirectly however, one can have as many arguments as one wants, including a variable number, which could for instance be a function of one of the arguments.)

```
\def\primes#1{2,~3%
  \n=#1 \advance\n by-2 %
  \p=5 %
  \loop\ifnum\n>0 \printifprime
      \advance\p by2 \repeat}
```

When the `\primes` macro is called, for instance with 30, `\primes{30}` is replaced by the body of `\primes` (that is, the group between braces which follows the list of `\primes`' formal arguments), in which `#1` is replaced by the two characters 3 and 0. `\primes{30}` hence becomes (we have removed spaces at the beginning of the lines, because they are ignored by TEX):

```
2,~3%
\n=30 \advance\n by-2 %
\p=5 %
\loop\ifnum\n>0 \printifprime
    \advance\p by2 \repeat
```

What happens now? We print "`2,~3`", that is, `2` followed by a comma, followed by an unbreakable space (i.e., the line will *in no case* be split after the comma); then 30 is assigned to `\n`. Immediately, 2 is subtracted from `\n`, and `\n` then contains the number of primes that remain to be printed. To keep it simple, we have assumed that at least the three first primes must be displayed. Therefore, we are sure that `\n` is at least equal to 1. This is also why it was possible to put a comma between `2` and `3`, because we know that 3 is not the last number to be printed. We want the last number printed to be preceded by "`and`". Hence, when we ask `\primes{3}`, we want to obtain “`2, 3, and 5`”. It should also be noticed that the “`%`” after “`3`” is essential to prevent insertion of a spurious space. “`3`” will be followed by a comma when `\printp` is called. The “`%`” after the second and third lines are not really needed since TEX gobbles all spaces after explicit numbers; these “`%`” signs appear only as remnants of comments.

We said that `\p` is the current number whose primality must be tested. We must therefore initialize `\p` to 5, since it is the first odd number after 3 (which we don't bother to check if it is prime or not).

The body of `\primes{30}` ends with a loop:

```
\loop\ifnum\n>0 \printifprime
    \advance\p by2 \repeat
```

It is a `\loop`/`\repeat` loop. In general, these loops have the form

```
\loop A text \if...   B text \repeat
```

This loop executes as follows: it starts with `\loop`, the A text is executed, then the `\if...` test. If this test succeeds, the B text is executed, then `\repeat` makes us return to `\loop`. If the test fails, the loop is over.

Hence, in the case of `\primes{30}`, it amounts to execute

```
\printifprime\advance\p by2
```

as long as `\n` is strictly positive, that is, as long as prime numbers remain to be printed. In order for this to produce the expected result, it is of course necessary to decrement the value of `\n`. This is done every time a number is printed with the call to `\printifprime`.

As a consequence, if at least one number remains to be printed, `\printifprime` will be called and will print `\p` if `\p` is prime. Whatever the result, we pass then to the next odd number with `\advance\p by2`.

Printing

The prime numbers are printed with `\printp`:

```
\def\printp{, %
    \ifnum\n=1 and~\fi
    \number\p \advance\n by -1 }
```

This macro is called only when `\p` is prime (see its call in `\printifprime`). In any case, this macro has no arguments and gets expanded into

```
, %
\ifnum\n=1 and~\fi
\number\p \advance\n by -1
```

that is a comma and a space, followed by “`and `” if `\n` equals 1 (in the case where the number to be printed is the last one), followed by `\p` (the `\number` function is analogous to `\the` and converts a variable into a sequence of printable characters);

finally, `\n` is decremented by 1, as announced, and this allows a normal unfolding of the `\loop...\repeat` loop in the `\primes` macro.

The macro `\printifprime` is called by `\primes`. It calls the function computing the primality of `\p` and this determines if `\p` must be printed or not.

```
\def\printifprime{\testprimality
                  \ifprime\printp\fi}
```

As one can guess, the `\testprimality` macro sets the "`prime`" boolean to "`true`" or "`false`," or if one prefers, it makes the `\ifprime` test succeed or fail.

Primality test

The macro testing `\p`'s primality uses the classical algorithm where divisions are tried by numbers smaller than `\p`'s square root.

```
\def\testprimality{{\d=3 \global\primetrue
  \loop\trialdivision
    \ifunknown\advance\d by2 \repeat}}
```

This macro is more complex because it involves an additional "group," shown here by the braces. Therefore, when `\testprimality` is expanded, we are left with

```
{\d=3 \global\primetrue
 \loop\trialdivision
   \ifunknown\advance\d by2 \repeat}
```

meaning that what happens between the braces will be — when not otherwise specified — local to that group. This was not the case in the expansions seen previously.

Let us first ignore the group. What are we doing? 3 is first assigned to `\d` where `\d` is the divisor being tested. We will test 3, 5, 7, etc., in succession, and this will go on as long as it is not known for certain whether `\p` is prime or not. As soon as we know if `\p` is prime or composed, the "`unknown`" boolean will become false and the `\ifunknown` test will fail.

Now, let us look at this again: we start with `\d=3`; the default is to consider `\p` prime, hence the "`true`" value is given to the "`prime`" boolean. This is normally done with

```
\primetrue
```

but in our case, it would not be sufficient. Indeed, at the end of

```
{\d=3 \primetrue
 \loop\trialdivision
   \ifunknown\advance\d by2 \repeat}
```

all variables take again their former value, because the assignments are *local* to the group. But the "prime" boolean is used when the \ifprime... test is being done in \printifprime, which is called after \testprimality. The group must therefore be *transcended* and the assignment is coerced to be global. This is obtained with

```
\global\primetrue
```

The remainder is then obvious: an attempt is made to divide \p by \d, and this is the purpose of \trialdivision. If nothing more has been discovered, that is, if "unknown" is still "true", the value of the trial divisor is set to the next value with \advance\d by2. Sooner or later this process stops, as shown by the \trialdivision definition.

The additional group in \testprimality can now be explained. If the group is not introduced, the expansion of \primes{30} leads to

```
...
\loop\ifnum\n>0 \printifprime
  \advance\p by2 \repeat
```

Plain TeX defines \loop as follows:

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body\let\next\iterate
  \else\let\next\relax\fi \next}
```

Therefore, the initial text is expanded into

```
\def\body{\ifnum\n>0 \printifprime
  \advance\p by2 }\iterate
```

Hence, the \loop...\repeat construct becomes

```
\ifnum\n>0 \printifprime\advance\p by2
           \let\next\iterate
\else \let\next\relax\fi \next
```

If \n > 0, this leads to

```
\printifprime ...
\let\next\iterate \next
```

and hence to

```
\testprimality ...
\let\next\iterate \next
```

and to

```
... \loop\trialdivision
    \ifunknown\advance\d by2 \repeat ...
\let\next\iterate \next
```

Now, \iterate will call \body, but the \body definition called will be the one defined by the second (inner) \loop, and chaos will follow! This explains why a group has been introduced. The group keeps the inner \body definition away from the outer \loop construct, hence each \iterate call produces the appropriate result.

Division trials

The last macro is where the actual division of `\p` by `\d` is made. An auxiliary variable `\a` is used.

```
\def\trialdivision{\a=\p \divide\a by\d
  \ifnum\a>\d \unknowntrue
  \else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse
              \unknownfalse\fi}
```

`\p` is copied into `\a`, then `\a` is divided by `\d`. This puts into `\a` the *integer part* of $\frac{\backslash p}{\backslash d}$. Two cases must then be considered:

1. if `\a` > `\d`, that is, if `\d` is smaller than the square root of `\p`, we are still in unknown territory. `\d` may be a divisor of `\p`, or there might be another divisor of `\p` larger than `\d` and smaller than the square root of `\p` root. The “`unknown`” boolean is therefore set to “`true`” with `\unknowntrue`.
2. if `\a` ≤ `\d`, we assume that we know, or at least, that we will know momentarily. We write therefore `\unknownfalse`.

In order to be sure, we must check if there is a remainder to `\p`’s division by `\d`, or rather to `\a`’s division by `\d`: `\a` is therefore multiplied by `\d`:

```
\multiply\a by\d
\ifnum\a=\p \global\primefalse
            \unknownfalse\fi
```

If `\p` is found again, it means that `\d` is one of `\p`’s divisors. In that case, `\p` is of course not prime and the “`prime`” boolean is set to false with `\primefalse`. Since `\trialdivision` is actually located in the group surrounding the body of the `\testprimality` macro, and since the “`prime`” is needed outside `\testprimality`, the group must once again be transcended and the “`prime`” assignment must be forced to be global. Hence:

```
\global\primefalse
```

Finally, in the case where `\d` divides `\p`, we set `\unknownfalse`, which has the sole effect of causing the loop to end:

```
\loop\trialdivision
  \ifunknown\advance\d by2 \repeat
```

that is, no other divisor is tested. One can observe that there is no `\global` in front of `\unknownfalse`, because `\ifunknown` is used within and not outside the group.

If `\p` is not found again after the multiplication, it means that `\d` is not a divisor of `\p`. At that time, we had

- either `\a` ≤ `\d`, and therefore `\a` < `\d` (otherwise `\p` would have been found after the multiplication), and hence `\unknownfalse`, therefore the loop

```
\loop\trialdivision
  \ifunknown\advance\d by2 \repeat
```
stops and since this happens in the context
```
\d=3 \global\primetrue
\loop\trialdivision
  \ifunknown\advance\d by2 \repeat
```
where "`prime`" had been set to true, we conclude naturally that, no divisor having been found up to `\p`'s square root, `\p` is prime.
Therefore, at the end of `\testprimality`'s call, `\ifprime` succeeds and `\p` is printed.

- or `\a` > `\d`: in that case, we know nothing more, `\unknowntrue`, and the next divisor must be tried.

Conclusion

This ends the explanation of these macros, apart from a few subtleties which were not mentioned.

It takes TEX a lot of time to do complex operations such as the ones described. In order to execute`\primes{30}`, TEX spends more time than it needs on average to typeset a whole page with plain TEX. `\trialdivision` is expanded 132 times. With `\primes{1000}` there are 41331 expansions and with `\primes{10000}` there are 1441624 expansions.

It should be stressed that the previous macros are given in *The TEXbook* [1, pp. 218–219], with the following lines as the only explanation:

> The computation is fairly straightforward, except that it involves a loop inside a loop; therefore `\testprimality` introduces an extra set of braces, to keep the inner loop control from interfering with the outer loop. The braces make it necessary to say '`\global`' when `\ifprime` is being set true or false. TEX spent more time constructing that sentence than it usually spends on an entire page; the `\trialdivision` macro was expanded 132 times.

TEX's programming language is quite peculiar and we gave only a glimpse of it. The interested reader should dive into TEX's "bible", namely Donald Knuth's *TEXbook* [1].

Acknowledgments

I would like to thank an anonymous referee for noticing an important error in the French version of the article.

References

[1] Knuth, Donald Ervin. *The TEXbook.* (Computers and Typesetting, Volume A). Reading, Massachusetts: Addison-Wesley, 1984. ISBN 0-201-13448-9.

Summary: Anatomy of a macro (tutorial)

In this article, we explain in detail a TEX macro computing prime numbers. This gives us an opportunity to illustrate technical aspects often ignored by beginners in the TEX world.

The source codes are included as small parts in the article commented in detail. You may find the original English version of the article in *TUGboat*, see `http://www.tug.org/TUGboat/Articles/tb22-1-2/tb70roeg.pdf`.

Keywords: Plain TEX, prime numbers.

Denis Roegel, `roegel@loria.fr`
`http://www.loria.fr/~roegel`
LORIA – Campus Scientifique, BP 239
F-54506 Vandœuvre-lès-Nancy Cedex, France

TEX Reference Card

(for Plain TEX)

Greek Letters

α	\alpha	ι	\iota	ϱ	\varrho
β	\beta	κ	\kappa	σ	\sigma
γ	\gamma	λ	\lambda	ς	\varsigma
δ	\delta	μ	\mu	τ	\tau
ϵ	\epsilon	ν	\nu	υ	\upsilon
ε	\varepsilon	ξ	\xi	ϕ	\phi
ζ	\zeta	o	\o	φ	\varphi
η	\eta	π	\pi	χ	\chi
θ	\theta	ϖ	\varpi	ψ	\psi
ϑ	\vartheta	ρ	\rho	ω	\omega
Γ	\Gamma	Ξ	\Xi	Φ	\Phi
Δ	\Delta	Π	\Pi	Ψ	\Psi
Θ	\Theta	Σ	\Sigma	Ω	\Omega
Λ	\Lambda	Υ	\Upsilon		

Symbols of Type Ord

<table>
<tr><td>$\aleph$</td><td>\aleph</td><td>$\prime$</td><td>\prime</td><td>$\forall$</td><td>\forall</td></tr>
<tr><td>$\hbar$</td><td>\hbar</td><td>$\emptyset$</td><td>\emptyset</td><td>$\exists$</td><td>\exists</td></tr>
<tr><td>$\imath$</td><td>\imath</td><td>∇</td><td>\nabla</td><td>$\neg$</td><td>\neg or \lnot</td></tr>
<tr><td>$\jmath$</td><td>\jmath</td><td>$\surd$</td><td>\surd</td><td>$\flat$</td><td>\flat</td></tr>
<tr><td>ℓ</td><td>\ell</td><td>$\top$</td><td>\top</td><td>$\natural$</td><td>\natural</td></tr>
<tr><td>$\wp$</td><td>\wp</td><td>$\bot$</td><td>\bot</td><td>$\sharp$</td><td>\sharp</td></tr>
<tr><td>$\Re$</td><td>\Re</td><td>$\|$</td><td>\|</td><td>$\clubsuit$</td><td>\clubsuit</td></tr>
<tr><td>$\Im$</td><td>\Im</td><td>$\angle$</td><td>\angle</td><td>$\diamondsuit$</td><td>\diamondsuit</td></tr>
<tr><td>∂</td><td>\partial</td><td>$\triangle$</td><td>\triangle</td><td>$\heartsuit$</td><td>\heartsuit</td></tr>
<tr><td>∞</td><td>\infty</td><td>$\backslash$</td><td>\backslash</td><td>$\spadesuit$</td><td>\spadesuit</td></tr>
</table>

Large Operators

$\sum$	\sum	$\bigcap$	\bigcap	$\bigodot$	\bigodot
$\prod$	\prod	$\bigcup$	\bigcup	$\bigotimes$	\bigotimes
$\coprod$	\coprod	$\bigsqcup$	\bigsqcup	$\bigoplus$	\bigoplus
$\int$	\int	$\bigvee$	\bigvee	$\biguplus$	\biguplus
$\oint$	\oint	$\bigwedge$	\bigwedge		

Binary Operations

$\pm$	\pm	$\cap$	\cap	$\vee$	\vee or \lor
$\mp$	\mp	$\cup$	\cup	$\wedge$	\wedge or \land
$\setminus$	\setminus	$\uplus$	\uplus	$\oplus$	\oplus
$\cdot$	\cdot	$\sqcap$	\sqcap	$\ominus$	\ominus
$\times$	\times	$\sqcup$	\sqcup	$\otimes$	\otimes
$\ast$	\ast	$\triangleleft$	\triangleleft	$\oslash$	\oslash
$\star$	\star	$\triangleright$	\triangleright	$\odot$	\odot
$\diamond$	\diamond	$\wr$	\wr	$\dagger$	\dagger
$\circ$	\circ	$\bigcirc$	\bigcirc	$\ddagger$	\ddagger
$\bullet$	\bullet	$\bigtriangleup$	\bigtriangleup	$\amalg$	\amalg
$\div$	\div	$\bigtriangledown$	\bigtriangledown		

Page Layout

\hsize=⟨dimen⟩	set width of page
\vsize=⟨dimen⟩	set height of page
\displaywidth=⟨dimen⟩	set width of math displays
\hoffset=⟨dimen⟩	move page horizontally
\voffset=⟨dimen⟩	move page vertically

Relations

$\leq$	\leq or \le	$\geq$	\geq or \ge	$\equiv$	\equiv
$\prec$	\prec	$\succ$	\succ	$\sim$	\sim
$\preceq$	\preceq	$\succeq$	\succeq	$\simeq$	\simeq
$\ll$	\ll	$\gg$	\gg	$\asymp$	\asymp
$\subset$	\subset	$\supset$	\supset	$\approx$	\approx
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\cong$	\cong
$\sqsubseteq$	\sqsubseteq	$\sqsupseteq$	\sqsupseteq	$\bowtie$	\bowtie
$\in$	\in	$\notin$	\notin	$\ni$	\ni or \owns
$\vdash$	\vdash	$\dashv$	\dashv	$\models$	\models
$\smile$	\smile	$\mid$	\mid	$\doteq$	\doteq
$\frown$	\frown	$\parallel$	\parallel	$\perp$	\perp
$\propto$	\propto				

Most relations can be negated by prefixing them with \not.

$\not\equiv$	\not\equiv	$\notin$	\notin	$\ne$	\ne

Arrows

$\leftarrow$	\leftarrow or \gets	$\longleftarrow$	\longleftarrow
$\Leftarrow$	\Leftarrow	$\Longleftarrow$	\Longleftarrow
$\rightarrow$	\rightarrow or \to	$\longrightarrow$	\longrightarrow
$\Rightarrow$	\Rightarrow	$\Longrightarrow$	\Longrightarrow
$\leftrightarrow$	\leftrightarrow	$\longleftrightarrow$	\longleftrightarrow
$\Leftrightarrow$	\Leftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow
$\mapsto$	\mapsto	$\longmapsto$	\longmapsto
$\hookleftarrow$	\hookleftarrow	$\hookrightarrow$	\hookrightarrow
$\uparrow$	\uparrow	$\Uparrow$	\Uparrow
$\downarrow$	\downarrow	$\Downarrow$	\Downarrow
$\updownarrow$	\updownarrow	$\Updownarrow$	\Updownarrow
$\nearrow$	\nearrow	$\searrow$	\searrow
$\nwarrow$	\nwarrow	$\swarrow$	\swarrow

The \buildrel macro puts one symbol over another. The format is \buildrel⟨superscript⟩\over⟨relation⟩.

$\stackrel{\alpha\beta}{\longrightarrow}$	\buildrel\alpha\beta\over\longrightarrow
$f(x) \stackrel{\rm def}{=} x+1$	f(x)\; {\buildrel\rm def\over=} \;x+1

Delimeters

<table>
<tr><td>$[$</td><td>\lbrack or [</td><td>$\{$</td><td>\lbrace or \{</td><td>$\langle$</td><td>\langle</td></tr>
<tr><td>$]$</td><td>\rbrack or]</td><td>$\}$</td><td>\rbrace or \}</td><td>$\rangle$</td><td>\rangle</td></tr>
<tr><td>$\vert$</td><td>\vert or |</td><td>$\lfloor$</td><td>\lfloor</td><td>$\lceil$</td><td>\lceil</td></tr>
<tr><td>$\Vert$</td><td>\Vert or \|</td><td>$\rfloor$</td><td>\rfloor</td><td>$\rceil$</td><td>\rceil</td></tr>
<tr><td>$[\![$</td><td>[\![</td><td>$(\!($</td><td>(\!(</td><td>$\langle\!\langle$</td><td>\langle\!\langle</td></tr>
<tr><td>$]\!]$</td><td>]\!]</td><td>$)\!)$</td><td>)\!)</td><td>$\rangle\!\rangle$</td><td>\rangle\!\rangle</td></tr>
</table>

Left and right delimeters will be enlarged if they are prefixed with \left or \right. Each \left must have a matching \right, one of which may be an empty delimeter (\left. or \right.). To specify a particular size, use the following:

\bigl, \bigr   \Bigl, \Bigr   \biggl, \biggr

You can also say \bigm for a large delimenter in the middle of a formula, or just \big for one that acts as an ordinary symbol.

Every Time Insertions

\everypar	insert whenever a paragraph begins
\everymath	insert whenever math in text begins
\everydisplay	insert whenever displayed math begins
\everycr	insert after every \cr

Accents

Type	Example	In Math	In Text
hat	$\hat{a}$	\hat	\^
expanding hat	$\widehat{abc}$	\widehat	none
check	$\check{a}$	\check	\v
tilde	$\tilde{a}$	\tilde	\~
expanding tilde	$\widetilde{abc}$	\widetilde	none
acute	$\acute{a}$	\acute	\'
grave	$\grave{a}$	\grave	\`
dot	$\dot{a}$	\dot	\.
double dot	$\ddot{a}$	\ddot	\"
breve	$\breve{a}$	\breve	\u
bar	$\bar{a}$	\bar	\=
vector	$\vec{a}$	\vec	none

The \skew⟨number⟩ command shifts accents for proper positioning, the larger the ⟨number⟩, the more right the shift. Compare

\hat{\hat A} gives $\hat{\hat{A}}$,   \skew6\hat{\hat A} gives $\skew6\hat{\hat{A}}$.

Elementary Math Control Sequences

overline a formula	$\overline{x+y}$	\overline{x+y}
underline a formula	$\underline{x+y}$	\underline{x+y}
square root	$\sqrt{x+2}$	\sqrt{x+2}
higher order roots	$\sqrt[n]{x+2}$	\root n\of{x+2}
fraction	${n+1 \over 3}$	{n+1\over 3}
fraction, no line	${n+1 \atop 3}$	{n+1\atop 3}
binomial coeff.	${n+1 \choose 3}$	{n+1\choose 3}
braced fraction	${n+1 \brace 3}$	{n+1\brace 3}
bracketed fraction	${n+1 \brack 3}$	{n+1\brack 3}

The following specify a style for typesetting formulas.

\displaystyle \textstyle \scriptstyle \scriptscriptstyle

Non-Italic Function Names

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

a \pmod{m}	$a \pmod{m}$	mod with parentheses
a \bmod m	$a \bmod m$	mod without parentheses

The following examples use \mathop to create function names.

Example	Command	Plain TEX Definition
$\lim_{x\to2}$	\lim_{x\to2}	\def\lim{\mathop{\rm lim}}
$\log_2$	\log_2	\def\log{\mathop{\rm log}\nolimits}

Footnotes, Insertions, and Underlines

\footnote⟨marker⟩{⟨text⟩}	footnote
\topinsert⟨vmode material⟩\endinsert	insert at top of page
\pageinsert⟨vmode material⟩\endinsert	insert on full page
\midinsert⟨vmode material⟩\endinsert	insert middle of page
\underbar{⟨text⟩}	underline text

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩

Useful Parameters and Conversions

\day,\month,\year	the current day, month, year
\jobname	name of current job
\romannumeral⟨number⟩	convert to lower case roman nums.
\uppercase{⟨token list⟩}	convert to upper case
\lowercase{⟨token list⟩}	convert to lower case

Fills, Leaders and Ellipses

Text or Math: ... \dots
Math: ... \ldots ⋯ \cdots ⋮ \vdots ⋱ \ddots

The following fill space with the indicated item.
\hrulefill \rightarrowfill \leftarrowfill \dotfill

The general format for constructing leaders is

\leaders⟨box or rule⟩\hskip⟨glue⟩	repeat box or rule
\leaders⟨box or rule⟩\hfill	fill space with box or rule

TeX Fonts and Magnification

\rm	Roman	\bf	**Bold**	\tt	Typewriter
\sl	*Slant*	\it	*Italic*	\/	"italic correction"

\magnification=⟨number⟩	scale document by $n/1000$
\magstep⟨number⟩	scaling factor of $1.2^n \times 1000$
\magstephalf	scalling factor of $\sqrt{1.2}$
\font\FN=⟨fontname⟩	load a font, naming it \FN
\font\FN=⟨fontname⟩ at ⟨dimen⟩	load font scaled to dimension
\font\FN=⟨fontname⟩ scaled ⟨number⟩	load font scaled by $n/1000$
true ⟨dimen⟩	dimension with no scaling

Alignment Displays

\settabs⟨number⟩\columns	set equally spaced tabs
\settabs\+⟨sample line⟩\cr	set tabs as per sample line
\+⟨text_1⟩&⟨text_2⟩&⋯\cr	tabbed text to be typeset
\halign	horizontal alignment
\halign to⟨dimen⟩	horizontal alignment
\openup⟨dimen⟩	add space between lines
\noalign{⟨vmode material⟩}	insert material after any \cr
\tabskip=⟨glue⟩	set glue at tab stops
\omit	omit the template for a column
\span	span two columns
\multispan⟨number⟩	span several columns
\hidewidth	ignore the width of an entry
\crcr	insert \cr if one is not present

Boxes

\hbox to⟨dimen⟩	hbox of given dimension
\vbox to⟨dimen⟩	vbox, bottom justified
\vtop to⟨dimen⟩	vbox, top justified
\vcenter to⟨dimen⟩	vbox, center justified (math only)
\rlap	right overlap material
\llap	left overlap material

Overfull Boxes

\hfuzz	allowable excess in hboxes
\vfuzz	allowable excess in vboxes
\overfullrule	width of overfull box marker. To eliminate entirely, set \overfullrule=0pt.

Indentation and Itemized Lists

\indent	indent
\noindent	do not indent
\parindent=⟨dimen⟩	set indentation of paragraphs
\displayindent=⟨dimen⟩	set indentation of math displays
\leftskip=⟨dimen⟩	skip space on left
\rightskip=⟨dimen⟩	skip space on right
\narrower	make paragraph narrower
\item{⟨label⟩}	singly indented itemized list
\itemitem{⟨label⟩}	doubly indented itemized list
\hangindent=⟨dimen⟩	hanging indentation for paragraph
\hangafter=⟨number⟩	start hanging indent after line n. If $n < 0$, indent first $\lvert n\rvert$ lines.
\parshape=⟨number⟩	general paragraph shaping macro

Headers, Footers, and Page Numbers

\nopagenumbers	turn off page numbering
\pageno	current page number. To get roman nums, set \pageno=⟨negative number⟩
\folio	current page number, roman num if < 0
\footline	material to put at foot of page
\headline	material to put at top of page. To leave space, set \voffset=2\baselineskip, make room with \advance\vsize by-\voffset.

Macro Definitions

\def\cs{⟨replacement text⟩}	define the macro \cs
\def\cs#1⋯#n{⟨repl. text⟩}	macro with parameters
\let\cs=⟨token⟩	give \cs token's current meaning

Advanced Macro Definition Commands

\long\def	macro whose args may include \par
\outer\def	macro not allowed inside definitions
\global\def or \gdef	definition that transcends grouping
\edef	expand while defining macro
\xdef or \global\edef	global version of \edef
\noexpand⟨token⟩	do not expand token
\expandafter⟨token⟩	expand item after token first
\futurelet\cs⟨tok_1⟩⟨tok_2⟩	equals \let\cs=⟨tok_2⟩⟨tok_1⟩⟨tok_2⟩
\csname...\endcsname	create a control sequence name
\string\cs	list characters in name, \ c s
\number⟨number⟩	list of characters in number
\the⟨internal quantity⟩	list of tokens giving value of quantity

Conditionals

The general format of a conditional is
\if⟨condition⟩⟨true text⟩\else⟨false text⟩\fi

\ifnum⟨num_1⟩⟨relation⟩⟨num_2⟩	compare two integers
\ifdim⟨dimen_1⟩⟨relation⟩⟨dimen_2⟩	compare two dimensions
\ifodd⟨num⟩	test for an odd integer
\ifmmode	test for math mode
\if⟨token_1⟩⟨token_2⟩	test if character codes agree
\ifdim	compare two dimensions
\ifx⟨token_1⟩⟨token_2⟩	test if tokens agree
\ifeof⟨number⟩	test for end of file
\iftrue, \iffalse	always true, always false
\ifcase⟨number⟩⟨text_0⟩\or⟨text_1⟩\or⋯ \or⟨text_n⟩\else⟨text⟩\fi	choose text by ⟨number⟩
\loop α \if...β \repeat	loop $\alpha\beta\alpha\cdots\alpha$ until \if is false
\newif\ifblob	create a new conditional called \ifblob
\blobtrue, \blobfalse	set conditional \ifblob true, false

Dimensions, Spacing, and Glue

Dimensions are specified as ⟨number⟩⟨unit of measure⟩.
Glue is specified as ⟨dimen⟩ plus⟨dimen⟩ minus⟨dimen⟩.

point	pt	pica	pc	inch	in	centimeter	cm
m width	em	x height	ex	math unit	mu	millimeter	mm
1 pc = 12 pt		1 in = 72.72 pt		2.54 cm = 1 in		18 mu = 1 em	

Horizontal Spacing: \quad (skip 1em) \qquad
Horizontal Spacing (Text): \thinspace \enspace \enskip \hskip⟨glue⟩ \hfil \hfill \hfilneg
Horizontal Spacing (Math): thin space \, medium space \> thick space \; neg. thin space \! \mskip⟨muglue⟩

Vertical Spacing: \vskip⟨glue⟩ \vfil \vfill

\strut	box w/ ht and depth of "(", zero width
	invisible box with dim of ⟨text⟩
\vphantom{⟨text⟩}	box w/ ht & depth of ⟨text⟩, zero width
\hphantom{⟨text⟩}	box w/ width of ⟨text⟩, zero ht & depth
\smash{⟨text⟩}	typeset ⟨text⟩, set ht & depth to zero
\raise⟨dimen⟩\hbox{⟨text⟩}	raise box up
\lower⟨dimen⟩\hbox{⟨text⟩}	lower box down
\moveleft⟨dimen⟩\vbox{⟨text⟩}	move box left
\moveright⟨dimen⟩\vbox{⟨text⟩}	move box right

Skip Space Between Lines:	\smallskip \medskip \bigskip
encourage a break	\smallbreak \medbreak \bigbreak
break if no room	\filbreak
Set Line Spacing:	\baselineskip = ⟨glue⟩
single space	\baselineskip = 12pt
1 1/2 space	\baselineskip = 18pt
double space	\baselineskip = 24pt
Increase Line Spacing	\openup⟨dimen⟩
use \jot's	1\jot = 3pt
Allow Unjustified Lines	\raggedright
Allow Unjustified Pages	\raggedbottom

Braces and Matrices

\matrix	rectangular array of entries
\pmatrix	matrix with parentheses
\bordermatrix	matrix with labels on top and left
\overbrace	overbrace, may be superscripted
\underbrace	underbrace, may be subscripted

For small matrices in text, use the following constructions:

{a\,b \choose c\,d}	${a\,b \choose c\,d}$
\left({a\atop c} {b\atop d} \right)	$\left({a\atop c} {b\atop d} \right)$

Displayed Equations

\eqno	equation number at right
\leqno	equation number at left
\eqalign	display several aligned equations
\eqalignno	display aligned equations numbered at right
\leqalignno	display aligned equations numbered at left
\displaylines	display several equations, centered
\cases	case by case definitions
\noalign	to insert space between lines in displays, use \noalign{\vskip⟨glue⟩} after any \cr
\openup⟨dimen⟩	add space between all lines in a display

Published by Ford & Mason Ltd, GL19 3JB, UK. Further copies of this card can be ordered through our web site: http://www.refcards.com.