

■ `\string(token)`: \TeX 首先读入 $\langle token \rangle$ 而不展开。如果控制系列记号出现, 那么它的 `\string` 展开由控制系列的名字组成(如果控制系列不单单是活动字符, 就把 `\escapechar` 包括进来作为转义符)。否则, $\langle token \rangle$ 就是字符记号, 并且其字符代码保持为展开后的结果。

■ `\jobname`: 展开为 \TeX 为此进程选定的名字。例如, 如果 \TeX 把它的输出放在文件 `paper.dvi` 和 `paper.log` 中, 那么 `\jobname` 就展开为 `'paper'`。

■ `\fontname(font)`: 展开为对应于所给定字体的外部文件名; 比如, `'\fontname\tenrm'` 将展开为 `'cmr10'`(五个记号)。如果字体所用的不是其设计尺寸, 那么“at size”也出现在展开中。 $\langle font \rangle$ 是一个由 `\font` 定义的标识符; 或是 `\textfont(number)`, `\scriptfont(number)` 或 `\scriptscriptfont(number)`; 或者是 `\font`, 它表示当前字体。

■ `\meaning(token)`: \TeX 把它展开为一系列字符, 它们是命令 `'\let\test=\langle token \rangle \show\test'` 在你的终端上显示的内容。例如, `'\meaning A'` 一般展开的是 `'the letter A'`; 在 `'\def\A#1B{C}'` 后, `'\meaning\A'` 展开的是 `'macro:#1B->C '`。

■ `\csname...\endcsname`: 当 \TeX 展开 `\csname`, 它要读入匹配的 `\endcsname`, 如果需要就展开记号; 在此展开中, 只有字符记号或保留下来。因此, 整个 `\csname...\endcsname` 文本的“展开”将是单个控制系列记号, 如果它当前没有定义, 则定义为 `\relax`。

■ `\expandafter(token)`: \TeX 首先读入 `\expandafter` 紧后面的记号, 而不展开它; 我们称此记号为 t 。接着, \TeX 读入 t 后面的记号(如果此记号有一个变量, 那么可能有更多的记号), 把它用其展开代替。最后 \TeX 把 t 放在此展开的前面。

■ `\noexpand(token)`: 展开为记号自己; 但是如果此记号是一个按 \TeX 的展开规则一般要被展开的控制系列, 那么其含义与 `'\relax'` 一样。

■ `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, 和 `\splitbotmark`: 展开为相应“标记”寄存器中的记号列(见第二十三章)。

■ `\input(file name)`: 展开为空; 但是 \TeX 做好准备从给定文件的读入内容, 之后再在当前文件中继续读入其它记号。

■ `\endinput`: 展开为空。 \TeX 到达了 `\input` 行的结尾, 将停止从包含此行的文件中读入。

■ `\the(internal quantity)`: 展开为一列记号, 它表示某个 \TeX 变量的当前值, 讨论见下面。例如, `'\the\skip5'` 将展开为 `'5.0pt plus 2.0fil'`(17 个记号)。



`\the` 这个有用的命令有很多子情形, 因此我们要同时讨论它们。各种内部数字量都可以被提出来使用:

■ `\the(parameter)`, 其中 $\langle parameter \rangle$ 是某个 \TeX 整数参数(比如, `\the\widowpenalty`), 尺寸参数(比如, `\the\parindent`), 粘连参数(比如, `\the\leftskip`), 或 `\muglue` 参数(比如, `\the\thinmuskip`)的名字。

■ `\the(register)`, 其中 $\langle register \rangle$ 是某个 \TeX 的整数寄存器(比如, `\the\count0`), 尺寸寄存器(比如, `\the\dimen169`), 粘连寄存器(比如, `\the\skip255`), 或 `\muglue` 寄存器(比如, `\the\muskip\count2`)的名字。

■ `\the<codename><8-bit number>`, 其中, `<codename>` 表示 `\catcode`, `\mathcode`, `\lccode`, `\uccode`, `\sfcode` 或 `\delcode`。例如, `\the\mathcode‘/’` 得到的是斜线的当前数学代码(整数)。


■ `\the<special register>`, 其中, 特殊寄存器为某个整数量 `\prevgraf`, `\deadcycles`, `\insertpenalties`, `\inputlineno`, `\badness` 或 `\parshape`(它只表示 `\parshape` 的行的数目); 或者是某个尺寸 `\pagetotal`, `\pagegoal`, `\pagestretch`, `\pagefilstretch`, `\pagefillstretch`, `\pagefilllstretch`, `\pageshrink`, `\pagedepth`。在水平模式下还可以指向一个特殊整数 `\the\spacefactor`, 在垂直模式下可用于一个特殊尺寸 `\the\prevdepth`。


■ `\the\fontdimen<parameter number>`, 它得到的是一个尺寸; 例如, 字体的参数 6 为其“em”的值, 因此, `\the\fontdimen6\tenrm` 得到的是‘10.0pt’(6 个记号)。

■ `\the\hyphenchar`, `\the\skewchar`, 它们得到的是定义给定字体的相应整数值。

■ `\the\lastpenalty`, `\the\lastkern`, `\the\lastskip`, 它们得到的是当前列中最后一个项目的 penalty, kerning, 粘连或 muglue 的量, 如果此项目分别是 penalty, kern, 或粘连的话; 否则得到的是‘0’或‘0.0pt’。


■ `\the<defined character>`, 其中 `<defined character>` 是一个控制系列, 它已经由 `\chardef` 或 `\mathchardef` 给定一个整数值; 结果就是此整数值, 用小数表示。

 在到现在为止的所有情况下, `\the` 得到的结果是一系列 ASCII 字符记号。除了字符代码为 32 的记号(“空格”)的类代码为 10 外, 每个记号的类代码都是 12 (“其它字符”)。同样的类代码规则也适用于由 `\number`, `\romannumeral`, `\string`, `\meaning`, `\jobname` 和 `\fontname` 得到的记号。


 在一些情形下, `\the` 得到的是非字符的记号, 是象 `\tenrm` 这样的字体标识符, 或者是任意记号列:

■ `\the` 得到的是选择给定字体的字体标识符。例如, `\the\font` 是对应于当前字体的控制系列。

■ `\the<token variable>` 得到的是一个变量当前值的记号列。例如, 可以展开 `\the\everypar` 和 `\the\toks5`。


 TeX 的原始命令 `\showthe` 将把在展开定义中 `\the` 所得到的东西显示在终端上; 展开前面加上 `>`, 后面跟上句点。例如, 如果采用 plain TeX 的段落缩进, 那么 `\showthe\parindent` 显示的是

`> 20.0pt.`

 下面是以前说过的可展开的记号不被展开的所有情形的列表。某些情形中含有未讨论过的原始命令, 但是我们最后会给出它们的。在下列情形下展开被禁止:

- 当记号在错误修复期间被删除时(见第六章)。
- 当因为条件文本被忽略而记号被跳过时。
- 当 TeX 正在读入宏的变量时。
- 当 TeX 正在读入由下列定义的控制系列时: `\let`, `\futurelet`, `\def`, `\gdef`, `\edef`, `\xdef`, `\chardef`, `\mathchardef`, `\countdef`, `\dimendef`, `\skipdef`, `\muskipdef`, `\toksdef`, `\read` 和 `\font`。

- 当 $\text{T}_{\text{E}}\text{X}$ 正在读入下列控制系列的变量记号时: `\expandafter`, `\noexpand`, `\string`, `\meaning`, `\let`, `\futurelet`, `\ifx`, `\show`, `\afterassignment`, `\aftergroup`。
- 当 $\text{T}_{\text{E}}\text{X}$ 正在读入的是 `\def`, `\gdef`, `\edef` 或 `\xdef` 的参数文本时。
- 当 $\text{T}_{\text{E}}\text{X}$ 正在读入的是 `\def` 或 `\gdef` 或 `\read` 的替换文本; 或者是象 `\everypar` 或 `\toks0` 这样的记号变量的文本; 或者是 `\uppercase` 或 `\lowercase` 或 `\write` 的记号列。(当 `\write` 的记号列实际输出到一个文件时要被展开。)
- 当 $\text{T}_{\text{E}}\text{X}$ 正在读入对齐的前言时, 但是除了在原始命令 `\span` 的一个记号后, 或者正在读入 `\tabskip` 的 `\glue` 时。
- 在数学模式开始的记号 $\$$ 紧后面时, 这是为了看看是否后面还跟着一个 $\$$ 。
- 在开始字母常数的记号 $'_{12}$ 的后面。


 有时候你会发现自己要定义一个新宏, 它的替换文本由于当前情形而已经被展开了, 而不是简单地逐字复制替换文本。为此, $\text{T}_{\text{E}}\text{X}$ 提供了命令 `\edef` (被展开的定义), 以及 `\xdef` (它等价于 `\global\edef`)。其一般格式与 `\def` 和 `\gdef` 一样, 但是 $\text{T}_{\text{E}}\text{X}$ 盲目地按照上面的展开规则来展开替换文本的记号。例如, 看看下面这个定义:


```
\def\double#1{#1#1}
\edef\a{\double{xy}} \edef\a{\double\a}
```

这里, 第一个 `\edef` 等价于 `\def\a{xyxy}`, 而第二个等价于 `\def\a{xyxyxyxy}`。所有其它类型的展开也要进行, 包括条件文本; 例如, 在 $\text{T}_{\text{E}}\text{X}$ 给出 `\edef` 时处在数学模式的情况下

```
\edef\b#1#2{\ifmmode#1\else#2\fi}
```

的结果等价于 `\def\b#1#2{#1}`, 否则结果等价于 `\def\b#1#2{#2}`。

 由 `\edef` 和 `\xdef` 给出的被展开的定义要把记号展开到只剩下不能展开的记号, 但是由 `\the` 生成的记号列不再进一步展开。还有, `\noexpand` 后面的记号不展开, 因为它的展开能力无效了。这两个命令可以用来控制展开哪些, 不展开哪些。

 例如, 假设你要把 `\a` 定义为 `\b` (展开的) 后面跟 `\c` (不展开) 再后面跟 `\d` (展开的), 并且假定 `\b` 和 `\d` 是无参数的简单宏。就可以用两种方法来实现:

```
\edef\a{\b\noexpand\c\d}
\toks0={\c} \edef\a{\b\the\toks0 \d}
```

甚至可以不用 `\noexpand` 或 `\the` 也可得到同样的效果; 对想多学习一些 $\text{T}_{\text{E}}\text{X}$ 展开原理的读者, 建议做一下下面三个练习。

练习20.14


不用 `\noexpand` 和 `\the`, 找出定义前一段中 `\a` 的方法。

练习20.15

继续讨论避免展开的例子, 假设要把 `\b` 完全展开到只剩下不能展开的记号, 但是你根本不希望展开 `\c`, 并且要把 `\d` 只展开一层。例如, 在定义 `\def\b{\c\c}`, `\def\c{*}` 和 `\def\d{\b\c}` 后, 要得到的是 `\def\a{**\c\b\c}` 这样的结果。怎样才能用 `\the` 来实现这个部分展开?

练习20.16

不用 `\the` 或 `\noexpand` 来解决上一个练习。(这个练习很难。)


 \TeX 的原始命令 `\mark{...}`, `\message{...}`, `\errmessage{...}`, `\special{...}` 和 `\write(number){...}` 都展开大括号中的记号列, 同 `\edef` 和 `\xdef` 基本一样。但是象 `#` 这样的宏参数字符在这样的命令中不用重复; 在 `\edef` 中用 `##`, 而在 `\mark` 中只用 `#`。命令 `\write` 有点特殊, 因为它的记号列首先读入而不展开; 直到记号实际上被写入一个文件时才进行展开。

练习20.17

比较下面两个定义:

```
\def\a{\iftrue{\else}\fi}
\edef\b{\iftrue{\else}\fi}
```

哪个得到未匹配的左括号? (有点技巧。)

 \TeX 可以同时从大约 16 个文件中读入各个文本行, 除了在 `\input` 后的文件之外。为了开始读入这样一个辅助文件, 应该使用

```
\openin(number)=<file name>
```


其中 `<number>` 在 0 和 15 之间。(Plain \TeX 用命令 `\newread` 了分配输入流的数字 0 到 15, 它类似于 `\newbox`。)在大多数 \TeX 的安装时, 如果没有明确给出扩展名, 那么扩展名 `‘.tex’` 将添加到文件名之后, 就象用 `\input` 一样。如果文件找不到, \TeX 不给出错误信息; 它只把输入流看作没有打开, 并且你可以用 `\ifeof` 来测试这种状态。当不再使用某个文件时, 可以使用

```
\closein(number)
```

并且如果与输入流数字相对应的文件是打开的, 那么它将关闭, 即, 返回其初始状态。为了从一个打开的文件得到输入, 可以使用

```
\read(number)to<control sequence>
```

并且所定义的控制系列是一个无参数的宏, 其替换文本是从指定文件读入的下一行的内容。此行用第八章的程序按当前类代码转换为一个记号列。如果需要, 再读入其它行, 直到左右大括号的数目相同。 \TeX 在要读入的文件结尾暗中添加了一个空行。如果 `<number>` 不在 0 和 15 之间, 或者如果这样的文件没有打开, 或者文件结束了, 那么输入就来自终端; 如果 `<number>` 不是负值, 那么 \TeX 将给出提示符。如果你不使用 `\global\read`, 那么宏的定义将是局域的。

 例如, 通过命令 `\read` 以及 `\message` (它把一个展开的记号列写在终端上和 log 文件中), 可以很容易实现与用户对话:

```
\message{Please type your name:}
\read16 to\myname
\message{Hello, \myname!}
```


在这种情况下, 命令 `\read` 将写入 `'\myname='` 并等待应答; 应答在 log 文件中重复出来。如果 `'\read16'` 变成 `'\read-1'`, 那么 `'\myname='` 就被省略了。


练习20.18

刚刚给出的 `\myname` 例子效果并不好, 因为在行尾的 `\return` 被转换为一个空格。看看怎样解决这个小问题?

练习20.19

继续看前一个例子, 定义一个叫 `\MYNAME` 的宏, 它包含所有 `\myname` 的字母, 但是用的是大写字母。例如, 如果 `\myname` 展开为 `Arthur`, 那么 `\MYNAME` 展开就是 `ARTHUR`。假定在 `\myname` 的展开中只包含字母和空格。

 附录 B, D, E 包含大量编写的宏的例子, 可以做很多事情。现在, 在本章结尾, 我们通过几个例子来说明作为编程语言, `TEX` 实际上可以怎样使用, 如果你要得到某些特殊的效果, 并且不在意计算机所耗的时间。

 Plain `TEX` 中含有一个 `\loop... \repeat` 结构, 它象这样工作: 给出 `'\loop α \if... β \repeat'`, 其中 α 和 β 是任意系列的命令, 并且 `\if...` 是任意条件测试(无匹配的 `\fi`)。 `TEX` 就首先执行 α ; 接着如果条件为真, `TEX` 就执行 β , 并且再次从 α 开始重复整个过程。如果条件为假, 循环就停止。例如, 下面有一个小程序, 进行一段对话, 其中 `TEX` 等待用户输入 `'Yes'` 或 `'No'`:

```
\def\yes{Yes } \def\no{No } \newif\ifgarbage
\loop\message{Are you happy? }
\read-1 to\answer
\ifx\answer\yes\garbagefalse % the answer is Yes
\else\ifx\answer\no\garbagefalse % the answer is No
\else\garbagetrue\fi\fi % the answer is garbage
\ifgarbage\message{(Please type Yes or No.)}
\repeat
```

练习20.20

用 `\loop... \repeat` 原理了编写一个一般的 `\punishment` 宏, 它重复任意给定段落任意给定次。例如,

```
\punishment{I must not talk in class.}{100}
```

将得到练习 20.1 所要的结果。