

第七章 用户定制 L^AT_EX

在 L^AT_EX 中用户可以定义自己的命令和环境。然而这不可避免地要频繁用到 L^AT_EX 的记数器和长度，因此我们首先详细讨论一下这些对象，并说明如何使用它们。

§7.1 记数器

§7.1.1 L^AT_EX 记数器

L^AT_EX 管理着大量的记数器，在启动时给出它们的初始值，通过调用特定命令可以改变它们的值。这些记数器的绝大多数都与可以改变它们的命令有相同的名称：

part	chapter	paragraph	figure	enumi
	section	subparagraph	table	enumii
	subsection	page	footnote	enumiii
	subsubsection	equation	mpfootnote	enumiv

从名称上就可以知道大部分记数器的意义，不需要再解释了。记数器 `enumi` ... `enumiv` 相应的是四个层次的 `enumerate` 环境（4.3.4 节和 4.3.5 节），而记数器 `mpfootnote` 控制 `minipage` 环境中的脚注编号（4.10.4 节）。

除这些记数器外，还可能存在用 `\newtheorem` 命令创建的记数器，它也具有与 结构类型 参数值相同的名称（4.5 节）。由于在 70 页上的 `\newtheorem` 命令例子，本书中也包含 `theorem` 和 `axiom` 两个记数器。

记数器的值必须是整数，通常也是非负的。一条命令可以同时输出几个值：当前的 `\subsection` 命令就输出 7.1.1，在这种情况下是调用了多个记数器。例如，`\subsection` 命令会给 `subsection` 记数器增 1，并显示 `chapter`, `section` 和 `subsection` 记数器的值，中间用句号分开。同时，命令还会把 `subsubsection` 记数器设为零，即使它没有出现。

§7.1.2 用户自定义的记数器

用户可以用下面的命令创建自己的记数器：

`\newcounter{记数器名}[上级记数器]`

这里 `记数器名` 就是刚建立的记数器的名称。它可以是任一字母的组合，只要不与已存在的记数器名称相同就可以了。因此不能用列在上面的 L^AT_EX 记数器或者前面已经定义的记数器名称作为 `记数器名`。可省参数 `上级记数器` 是另一个已经存在的记数器（L^AT_EX 的或用户自定义的）的名称，其作用就在于只要 `上级记数器` 被 `\stepcounter` 或 `\refstepcounter` 命令（见下面）增 1，就把新建立的记数器重设为零。

当用 `\newcounter` 创建了一个新的计数器，它的初始值就是零。

`\newcounter` 计数器不能位于用 `\include` 命令（8.1.2 节）读进的文件中。因此最好把所有的 `\newcounter` 命令都放在导言中。

§7.1.3 改变计数器的值

无论是 *LT_EX* 计数器，还是用户自定义的计数器，都可以用下面的命令改变其值：

`\setcounter{计数器}{数}`

这条命令的意义从字面上就可以知道了：指定的计数器被赋予给定的数值（整数）。

`\addtocounter{计数器}{数}`

利用这条命令，指定计数器的值增加了给定数值，数可以是正值，也可以是负值。

`\stepcounter{计数器}`

指定计数器的值增 1，同时所有从属计数器（即所有把这个计数器作为自己上级计数器的计数器）的值被重设为零（见上）。

`\refstepcounter{计数器}`

这条命令的效果与 `\stepcounter` 相同，但它也同时把 counter 设为交叉索引命令 `\label` 中的当前计数器（见 8.3.1 节）。

例如，最后一条命令可以用在没有 `\caption` 命令 `figure` 或 `table` 环境中，这样也可以在正文中引用这些插图或表格的编号。那么放在浮动环境中的 `\refstepcounter{figure}` 或 `\refstepcounter{table}` 命令也可以使得相应的计数器变为正确的值，从而可以用 `\label` 命令给它赋予一个关键词（8.3.1 节）。

计数器的值可以用下面的命令当做一个数值处理：

`\value{计数器}`

这条命令并不改变计数器的值。它通常与 `\setcounter` 或 `\addtocounter` 结合使用。例如，若用户已经创建了计数器 `mypage`，那么就可以用命令

`\setcounter{mypage}{\value{page}}`

使它取与页码计数器 `page` 有相同的值。

通常 `\protect` 命令可以用来保护脆弱命令在传送过程被破坏，它同样也可以放在牢固命令前面，而不会有任何危害。然而 `\value` 是一个例外。从来不要为它前面加上 `\protect` 命令。

§7.1.4 显示计数器的值

在计数器中的值可以用下面的命令显示出来：

<code>\arabic{counter}</code>	以阿拉伯数字显示,
<code>\Roman{counter}</code>	以大写罗马数字显示,
<code>\roman{counter}</code>	以小写罗马数字显示,
<code>\alph{counter}</code>	以小写字母显示,
<code>\Alph{counter}</code>	以大写字母显示,
<code>\fnsymbol{counter}</code>	以脚注符号显示。

在命令 `\alph` 和 `\Alph` 中, 数字 1...26 对应着字母 a...z 和 A...Z。这就需要用户保证计数器的值位于这个范围里。对于 `\fnsymbol`, 数字 1...9 输出的符号分别是 * † ‡ § ¶ || ** †† ‡‡。这里也同样要求用户保证计数器的值不要达到或超过 10。

有一些计数器, 还存在下面这样形式的命令:

`\the` 计数器

这里 `\the` 紧接着 计数器 的名称, 如 `\thepage`。这种命令通常与 `\arabic{计数器}` 是一样的, 但也可以是几条计数器命令组成的。例如, 在文档类 `book` 和 `report` 中, 命令 `\thesection` 就是用章和节号组成的:

`\arabic{chapter}.\arabic{section}`

这里 `\thesection` 的结果就是 7.1。

页码、公式或章节编号等等的自动显示, 都是通过调用适当的 `\the` 计数器命令完成的。如果需要另一种不同格式的自动编号, 比如说字母型的公式编号, 相应 `\the` 计数器命令的定义就可以用 7.3 节中的方法进行修改。

练习 7.1: 使用标准练习文件 `exercise.tex`, 用 `\arabic{counter}` 在结尾出打印出 L^AT_EX 计数器的值。利用 `\setcounter` 和 `\addtocounter` 命令改变一些计数器的值, 然而再打印出结果。

§7.2 长度

我们在前面已经多次指出, 类似于 `\parskip` 或 `\textwidth` 这样的长度参数的值可以用命令 `\setlength` 来设成新值。有些参数需要取可以伸展和收缩的橡皮长度。这些参数主要用来生成竖直或水平距离。在 2.4 节中已详细给出了长度 (无论固定长度还是橡皮长度) 单位的类型。这里不会再重复, 在这一节中讨论其它的赋值和控制长度的命令。

给长度参数赋值的标准 L^AT_EX 方法是用下面的命令

`\setlength{\长度命令}{长度指定}`

这里 长度指定 可以是指定长度 (要有单位) 或者另一个长度参数。在后一种情形中, `\长度命令` 就取这个参数的当前值。因此在一个 `list` 环境中用 `\setlength{\rightmargin}{\leftmargin}` 就可以使得右页边与左页边相同。

可以用下面的命令增加长度值:

`\addtolength{\长度命令}{长度指定}`

这条命令就把 长度指定 加到 `\长度命令` 参数上去。若 长度指定 为负值, 就减去相应的量。同样, 可以用另一个长度参数作为 长度指定, 参数前面可以有负号, 这样就可以加上或减去这个参数。在长度参数前面的数值会与参数中的值相乘: `0.5\textwidth` 意味着文本列宽的一半, 而 `2\parskip` 是段间距的两倍。

利用命令

`\settowidth{\长度命令}{文本}`

可以使 `\长度命令` 参数的值等于处于 LR 模式 (通常是从左到右) 的一块文本 的自然宽度。

类似地, 命令

`\settoheight{\长度命令}{文本}`

`\settodepth{\长度命令}{文本}`

把 `\长度命令` 的值分别取为文本在基线上方或下方的高度与深度。

最后, 命令

`\stretch{小数}`

生成一个橡皮长度, 其可展性是 `\fill` 的给定 小数 倍 (2.4.2 节)。

用户要自己定义长度, 可以用如下命令:

`\newlength{\新长度命令}`

这样就可以建立起长度 `\新长度命令`, 初始值为 0pt。上面所讲的命令都可以用来处理它的值。

命令

`\addvspace{长度指定}`

会在其所处的地方插入给定 长度指定 的额外竖直距离。如果同时多次使用这条命令, 那么实际被插入的间距是其中最大的那个, 而不是所有间距的总和。这条命令只能用在两段之间。把它应用于用户自己定义的命令和环境中, 可以使得生成的结构更像段落。

§7.3 用户定义命令

在 *L^AT_EX* 中可以用下面的命令定义或重定义新的命令:

`\newcommand {\命令名称} [参数个数] [可省参数] {定义}`

`\renewcommand {\命令名称} [参数个数] [可省参数] {定义}`

或者

`\newcommand {\命令名称} [参数个数] {定义}`

2.09 `\renewcommand{命令名称}[参数个数]{定义}`

这两组中的第一条命令是用来定义不存在的新命令。命令名称可以是字母的任意组合，只要不与别的命令重名即可。第二条命令是用来重定义一条已存在的命令。对这两种情形，如果调用了不正确的变量，都会给出一条错误信息。第一个可省参数 参数个数 是一个介于 1 到 9 之间的数，它规定了新定义的命令或者被改变了定义的命令中有多少个参数值。在 \LaTeX 2_ϵ 中可以存在的第二个可省参数值 可省参数 给出了新命令可以为可省参数值取的默认值。命令的实际定义是包含在 定义 中。

§7.3.1 没有参数值的命令

我们首先演示没有可省参数值 参数个数 的 `\newcommand` 命令的用法。当一种固定的 \LaTeX 命令或用户命令组合被多次重复时，就可以用这种形式的命令给它赋一名称。例如，结构 x_1, \dots, x_n 称为 x - 向量，经常出现在数学公式中，它是用数学模式中的 `x_1, \ldots, x_n` 生成的。为此输入

```
\newcommand{\xvec}{x_1, \ldots, x_n}
```

就可以创建一个新的命令，名称为 `\xvec`。此后就可以同其它命令一样调用这条新定义的命令。当调用它时，它就在自己所处的地方插入文本或命令序列（即这里的 `x_1, \ldots, x_n`）。事实上，这里的过程是：当 `\xvec` 被调用时，它的定义就进入 \LaTeX 处理系统中。

由于新的 `\xvec` 命令定义中包含数学命令（下标命令 `_`），因此只能在数学模式中调用。从而在文本模式中需要用 `\$ \xvec \$` 来得到 x_1, \dots, x_n 。从这点来看，在定义中包含数学模式切换也不失为一个好主意：

```
\newcommand{\xvec}{\$x_1, \ldots, x_n\$}
```

这样 `\xvec` 就生成 x_1, \dots, x_n 。然而，这样一来，它就只能用在文本模式中，而不能再用在数学模式中了。下面是一种可以保证命令在两种模式中都可以使用的技巧：把命令定义为

2ε `\newcommand{\xvec}{\ensuremath{x_1, \ldots, x_n}}`

这样 `\xvec` 和 `\$ \xvec \$` 都是可以接受的，而且结果一样。

（在 \LaTeX 2.09 中，没有 `\ensuremath` 命令。此时在数学模式中我们可以用 `\mbox{\$...\$}`，因为在文本模式中，`\mbox` 是被忽略的，但在数学模式中，它就可以暂时切换到文本模式中，而其中的 `\$` 符号又可以激活数学模式。这两种方法得到的结果并不一样，相比之下，在 \LaTeX 2_ϵ 中的 `\ensuremath` 结果就要好得多。）

在文本中应用 `\xvec` 时应该写成 `\xvec{}`，这是因为 \TeX 认为它是一个没有参数值的命令，当它遇到第一个非字母字符时就终止其名称。如果这里遇到的第一个非字母字符是空格，那它就结束命令名称，但并不插入单词间隔（2.1 节）。因此 `\xvec and ...` 的结果是 ‘ x_1, \dots, x_n and ...’，其没有单词

间隔。只要在命令名称后面插入一个空格命令 `_` 或者空结构 `{ }` 就可以解决这个问题，因此这里可以用 `\xvec_` 和 `\xvec{ }`。

当然也可以在 `\xvec` 定义中就包含空格，即 `{\ensuremath{x_1,\ldots,x_n}}`。现在跟在命令名称后面的空格还是要被去掉的，但命令本身会插入一个空格。然而，在实际中我们不推荐这种做法，因为这个程序设计中的空格就总是存在的，即使接在命令后面的是标点符号或其它符号也不例外。

上面各个不同的例子中用的都是命令 `\newcommand`，但实际上这条命令只能用一次，以初始化用户定义的还不存在的命令。一旦 `\xvec` 已经被定义好，那么修正版本只能用 `\renewcommand` 命令来给出。实际上这里的第二个和第三个 `\xvec` 命令就是这样做的。

照这个例子的样子，用户可以通过 `\newcommand`(或者 `\renewcommand`) 命令来给命令和文本组合起一个新名称，然后在需要的时候调用它。利用这种方法，输入量会显著减少，而且也会减少出错的机会，特别是处理复杂数学结构的时候。

如果不确定给命令选择的名称是否已经存在，那么可以用

2ε `\providecommand{\命令名称}[参数个数][可省参数]{定义}`

这条命令的语法与 `\newcommand` 和 `\renewcommand` 命令完全一样。差别就在于如果命令已经存在，新定义就被忽略。想取得相反的效果（在不用确知命令是否存在的条件下，覆盖命令的当前定义）可以用如下方法得到：首先调用 `\providecommand` 确保命令存在，然后用 `\renewcommand` 命令给出真正的定义。然而，做这一操作时需要特别细心！

练习 7.2: 定义命令 `\iint`, `\iiint` 和 `\idotsint`,

生成如右所示的显示公式中的多重积分符号，或

或者是下面这样的正文公式： $\iint, \iiint, \int \cdots \int$ 。

$$\iint \iiint \int \cdots \int$$

练习 7.3: 修改 `\thechapter`, `\thesection` 和 `\thesubsection` 命令的定义，使得在 `book` 和 `report` 文档类中章节号以大写字母显示，例如 B，而节号是用大写罗马数字接在章号后面，形式为 B-III，而小节编号用的是小写罗马数字，与前面之间用逗号分开：B-III,v。

提示：在 `book` 和 `report` 类中，这些命令的原始定义为

```
\newcommand{\thechapter}{\arabic{chapter}}
\newcommand{\thesection}{\thechapter.\arabic{section}}
\newcommand{\thesubsection}{\thesection.\arabic{subsection}}
```

现在用 `\renewcommand` 命令进行必要的修改。

§7.3.2 有参数值的命令

除了结构 x_1, \dots, x_n 外，在数学中还有 y_1, \dots, y_n 和 z_1, \dots, z_n 等等同样

的向量结构。因此我们可以按 `\xvec` 的样子定义命令 `\yvec` 和 `\zvec`。然而，也可以用定义一个广义的向量命令，把变化部分作为参数值。对于当前这个例子，变化部分是字母 x, y 和 z 。只有一个变量的命令是用可省参数值 [1] 生成的。因此，

```
\newcommand{\avec}[1]{\ensuremath{\#1_1,\ldots,\#1_n}}
```

就定义了一般性的向量命令 `\avec{参数值}`。这样当调用 `\avec{x}` 时结果就是 x_1, \dots, x_n ，而 `\avec{y}` 显示出 y_1, \dots, y_n 。在命令定义中的字符 `#1` 只是一个哑元，表示当命令被调用时，要用 参数值 的文本取代所有出现的 `#1`。只要把定义中所出现的 `#1` 都想像成 x 或 y ，就可以相当容易地了解所定义的结构了。

在哑元 `#1` 中的数字 1 在这里好像没有什么用处。事实上，对于只有一个参数值的命令，它确实没有什么意义。然而，对于多参数值的命令，它的作用就很明显了。比如说，我们要定义一条命令，它既能生成结构 x_1, \dots, x_n ，也能生成 v_1, \dots, v_m 。这就需要有两个参数值，一个用来指定 u, v 等等，另一个用来确定最后的那一个下标 n, m ，等等。这样的命令是如下生成的：

```
\newcommand{\anvec}[2]{\ensuremath{\#1_1,\ldots,\#1_{\#2}}}
```

这样 `\anvec{u}{n}` 就得到 u_1, \dots, u_n ，而 `\anvec{v}{m}` 得到 v_1, \dots, v_m 。`\newcommand` 命令中的可省参数值 [2] 说明要定义的命令中包含两个参数值；在定义部分，`#1` 表示第一个参数值，`#2` 表示第二个参数值。把 `#1` 想像成 u 或 v ，把 `#2` 想像成 n 或 m ，那么就可以很容易知道 `\anvec{arg1}{arg2}` 的操作方式了。

这种方式可以适用于有更多的参数值。做了如下定义后，

```
\newcommand{\subvec}[3]{\ensuremath{\#1_{\#2},\ldots,\#1_{\#3}}}
```

命令 `\subvec` 就是有三个参数值的命令。从其定义易知 `\subvec{a}{i}{j}` 生成 a_i, \dots, a_j 。

只由单个字符构成的参数值不必放在大括号 `{ }` 内，可以直接给出。如果它是第一个参数值，那必须像通常那样，用空格把它与命令名分开。因此 `\subvec aik` 的结果与 `\subvec{a}{i}{k}` 的一样，而 `subvec x1n` 生成与前面例子 `\xvec` 同样的结构 x_1, \dots, x_n 。

当参数值不只由一个字符组成时，它就必须放在大括号 `{ }` 内，因为这里的大括号表示要把参数值内容当做一个整体处理。因此 `\subvec{A}{ij}{lk}` 的结果为 A_{ij}, \dots, A_{lk} 。这里的参数取代是 A 相应于 `#1`， ij 相应于 `#2`， lk 相应于 `#3`。

可是为什么 `\subvec{A}{ij}{lk}` 的结果是 A_{ij}, \dots, A_{lk} ，而不是所期望的 A_{ij}, \dots, A_{lk} 呢？这是因为虽然在大括号内的参数值被当做一个整体替换进定义文本中，但大括号自身并没有进去。这里替换后的命令文本实际上类似于 `\ensuremath{A_{ij},\ldots,A_{lk}}`，因此实际上只有接在下标符号 `_` 后面

的第一个字母被降低。为了使两个字母都被降低，那在命令文本中它们就必须做为一个整体出现，也就是类似于 A_{ij}, \dots, A_{lk} 。可以在 `\subvec` 命令的参数值中额外再加一对大括号就可以做到这一点，即 `\subvec{A}{\{ij\}}{\{lk\}}`。而更好的解决方法是一开始就在定义中加上大括号：

```
\newcommand{\subvec}[3]{\ensuremath{\#1_{\#2}, \ldots, \#1_{\#3}}}
```

这样即使每个参数值只有一层大括号，也能得到所期望的结果：

`\subvec{A}{ij}{lk}` 的结果为 A_{ij}, \dots, A_{lk} 。

§7.3.3 有一个可省参数值的命令 2_ε

我们已经知道有很多 L^AT_EX 命令可以有可省参数值，这其中最典型的例子就是 `\newcommand` 命令自身。在 L^AT_EX 2_ε 中，同样也可以使用用户定义的命令有一个可省参数值。这样做的好处就在于，虽然多提供了一个参数，但在绝大多数情况下它取得只是标准值，不需要显式改变其值。

例如，在上一节用户定义的命令 `\subvec` 中有三个参数值，分别相应于字母和第一个及最后一个下标。然而，通常字母就是 x ，因此把它做为一个可省参数不失为一个好主意，这样只有在字母不同时才需要指定。要做到这一点，利用如下命令：

```
\renewcommand{\subvec}[3][x]{\ensuremath{\#1_{\#2}, \ldots, \#1_{\#3}}}
```

它与前面定义的区别就在于 [3] 参数值后面多了 [x]。这就说明了三个参数值中的第一个是可省的，其标准值为 x 。现在 `\subvec{i}{j}` 的结果就是 x_i, \dots, x_j ，而 `\subvec[a]{1}{n}` 的结果为 a_1, \dots, a_n 。

在用户定义的命令中只能有一个可省参数，而且也必须是第一个，即定义中的 #1。

§7.3.4 用户定义命令的其它样例

在上面用户定义命令的解释中，我们用向量结构这样很简单的情形做为示例。下面我们要演示的是更复杂的情况，其中要应用到计数器、长度，甚至一些特殊的 T_EX 命令。

例1：在 5.4.6 节中，说明了 T_EX 命令 `\atop` 和 `\choose` 是相当有用的数学命令，即使在 L^AT_EX 中也不例外。不幸的是这两条命令的语法同类似的 L^AT_EX 命令 `\frac` 大相径庭。然而，

```
\newcommand{\latop}[2]{\#1\atop\ #2}
```

```
\newcommand{\lchoose}[2]{\#1\choose\ #2}
```

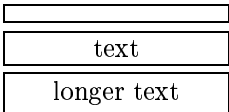
定义的两条命令 `\latop` 和 `\lchoose` 就生成同样的结果，而且语法也符合通常的 L^AT_EX 结构化要求：`\latop{上式}{下式}`。

例2：在下面我们要定义两条命令，`\defbox{取样文本}` 会设置一个盒子，其宽度等于 `取样文本` 的长度。然后调用 `\textbox{文本}` 命令，就会在一个宽

度与 取样文本 相同的有框盒子中居中显示 文本。

```
\newlength{\width}
\newcommand{\defbox}[1]{\settowidth{\width}{#1}}
\newcommand{\textbox}[1]{\framebox[\width]{#1}}
```

首先创建了一个新的长度参数 `\width`，然后定义 `\defbox`，使得 `\width` 就等于其参数值的长度（7.2 节），最后用 `\textbox` 生成一个相同宽度的有框盒子，其中文本居中。（不要用长度参数 `\width`，因为它已经存在，见 4.7.5 节。）

<pre>as wide as this text\\ \defbox{as wide as this text}\textbox{}\\ \textbox{text}\\ \textbox{longer text}</pre>	<pre>as wide as this text</pre> 
--	--

例3: 我们要定义脚注命令 `\myftnote`，它与通常的 `\footnote{文本}` 命令一样把 文本 放到脚注处，但这里不是用数字做脚注标记，而是依次用符号 * † ‡ § ¶ || ** †† ‡‡，在每一新页上都是从 * 开始。首先需要创建一个新的计数器，每当 `page` 计数器增 1 时，其都要被重置为零。做法为（见 7.1.2 节）：

```
\newcounter{myfn}[page]
```

这个用户自定义的计数器 `myfn` 每当 `page` 增 1 时都会自动重置为零。下面这条命令

```
\renewcommand{\thefootnote}{\fnsymbol{footnote}}
```

重定义脚注标记为计数器 `footnote` 所对应的上述符号（4.10.2 和 7.1.4 节）。现在可以如下构造新的脚注命令：

```
\newcommand{\myftnote}[1]{\setcounter{footnote}{\value{myfn}}%
\footnote{#1}\stepcounter{myfn}}
```

其就会生成所期望的结果。* 用户定义的命令 `\myftnote` 拥有一个参数值，当把 L^AT_EX 计数器 `footnote` 的值与用户定义计数器 `myfn` 相同值时，把它传送给 L^AT_EX 的 `\footnote` 命令。一旦执行了命令 `\footnote`，就用命令 `\stepcounter{myfn}` 给计数器 `myfn` 增 1。但是一旦 `page` 计数器增 1，即生成一个新页，这个计数器就要被重置为零。

上面的脚注就是用命令 `\myftnote` 生成的。同样现在也是用的这条命令，† 再用一次。‡ 这演示了所定义符号的使用。

例4: 我们下面定义命令 `\alpheqn`，一旦它被调用，后续公式将具有相同的编号，只是后缀字母 a, b, ...，中间用连字符 ‘-’ 分开。命令 `\reseteqn` 把

* 放在第一行结尾处的 % 符号是为了去掉行尾隐藏的空格被解释成命令的一部分（4.11 节）。通常为了增加可读性，需要把命令定义分成几行。

† 另一个脚注

‡ 还是一个脚注

编号框架重设为原来的样式。因此公式编号序列可能有如此形式： 4, 5, 6-a, 6-b, 7。

```
\newcounter{saveeqn}%
\newcommand{\alpheqn}{\setcounter{saveeqn}{\value{equation}}}%
\stepcounter{saveeqn}\setcounter{equation}{0}%
\renewcommand{\theequation}
    {\mbox{\arabic{saveeqn}-\alph{equation}}}%
\newcommand{\reseteqn}{\setcounter{equation}{\value{saveeqn}}}%
\renewcommand{\theequation}{\arabic{equation}}}%
```

由于我们在 7.1 节中已讲解了上面所用的命令和计数器，因此这个例子是很好理解的。计数器 `equation` 的当前值被保存到计数器 `saveeqn` 中，然后增加 `saveeqn`，而 `equation` 被重置为零。公式编号的形式 `\theequation` 利用这两个计数器进行了重定义。这样公式编号的方式就同原来一样，而 `saveeqn` 没有改变。重设命令 `\reseteqn` 就把 `saveeqn` 的值重新赋给 `equation`，并恢复 `\theequation` 的定义。

这条命令只适合用在 `article` 文档类中。如果所用文档类是 `book` 或者 `report`，那么 `\theequation` 的定义是

```
\arabic{chapter}.\arabic{equation}
```

这里需进行的必要修改，就留给读者做为练习。

在定义组合公式编号的第一个 `\renewcommand{\theequation}` 命令中的 `\mbox` 命令是必须的，因为结果要用数学模式显示，这样连字符 ‘-’ 要被解释成二元运算符（负号），其前后与两个被操作数 `\arabic{saveeqn}` 和 `\alph{equation}` 之间就会有额外的间距。因此结果可能是 $6 - a$ ，而不是 $6-a$ 。`\mbox` 命令就是为了暂时从数学模式切换进 LR 模式。

例：最后我们利用某些很基本 T_EX 命令，给出一个例子，这里无法详细解释这些命令。但是这里定义的命令对于那些输入文本中经常包含化学公式的情形是非常有用的。

在 5.4.9 节中指出在化学公式 $\text{Fe}_2^+\text{Cr}_2\text{O}_4$ 中的下标不在同一水平线上，而且在文本模式中输入相对短的公式也很烦。下面这条命令就可以解决这些问题。

```
\newlength{\fntxvi} \newlength{\fntxvii}
\newcommand{\chemical}[1]
{{\fontencoding{OMS}\fontfamily{cmss}\selectfont
    \fntxvi\the\fontdimen16\font
    \fntxvii\the\fontdimen17\font
    \fontdimen16\font=3pt\fontdimen17\font=3pt
    $\mathrm{#1}$}}
```

```
\fontencoding{OMS}\fontfamily{cmys}\selectfont
\fontdimen16\font=\fntxvi \fontdimen17\font=\fntxvii}}
```

现在 `\chemical{Fe_2^{2+}Cr_2O_4}` 的结果就是正确的形式 $\text{Fe}_2^{2+}\text{Cr}_2\text{O}_4$ 。

解释：TeX命令 `\fontdimen n` 描述了字符字体的特定性质，如 $n = 16$ 和 $n = 17$ 就确定指标（下标）的位置。在用 `\selectfont`（8.5.1 节讲到）结尾的那行上的命令使得当前数学符号字体连同其内部设计一起存贮在 TeX 命令 `\font` 中。当前尺寸（这里是 10pt）下数学公式中的所有指标都是一致降低 3.0pt。关于其它尺寸，请见练习 7.7 的提示。由于对 `\fontdimen` 的改变总是全局性的（当从一个真正的或者隐式的环境中退出时，并不恢复以前的值），因此有必要首先保存当前值，然后再手工恢复。

练习 7.4: 同例 1 中定义 `\lato` 和 `\lchoose` 一样的方式，定义 L^ATeX 命令 `\lbrack` 和 `\lbrace`，分别相应于 TeX 命令 `\brack` 和 `\brace`。这两条 TeX 命令的形式同 5.4.6 节的 `\choose` 一样，只是它们用中括号 `[\lbrack]` 或者大括号 `{\lbrace}` 把其中内容包围起来。

练习 7.5: 把例 4 中命令推广成 `\vareqn{数}{类型}`，使得后续公式的编号形式为 数 后接放在中括号内的活动编号，其中活动编号的显示由 类型 参数值确定。例如 `\vareqn{33}{\Alph}` 的结果可能为 33[A], 33[B] 等等。

练习 7.6: 推广练习 7.2 中的积分命令，引进一个参数值表示水平居中放在积分号下面的积分区域。因此 `\iint{(D)}`，`\iiint{V}` 和 `\idotsint{G}` 的结果应如右所示。

$$\iint_{(D)} \iiint_V \int \cdots \int_G$$

提示：第二条命令只要把下标放在中间那个积分号下面就可以了（见 5.2.5 节的 `\limits`）。而对于其它两个，下标符号需要利用 `\hspcae{-...}` 进行负的水平位移。

练习 7.7: 在上面例 5 中化学公式下标下沉 3pt 是相应于 10pt 的字体尺寸的。对于 11pt 和 12pt，这个值是 3.3 和 3.6pt 就比较恰当。推广命令 `\chemical`，使得这个值成为一个可省参数值，其默认值为 3pt。这就是说对于 12pt 的字体尺寸，调用方式应为 `\chemical[3.6pt]{...}`。

§7.3.5 条件文本

有经验的 TeX 用户对在 TeX 和 L^ATeX 中都可以用的条件文本是相当熟悉的。然而，这种用法并不是那么简单明了，需要对 TeX 深层机制有相当的了解。Leslie Lamport 提供了一个叫 `ifthen` 的软件包，David Carlisle 把它进行了推广，在 L^ATeX 2_ε 中也可以应用，这个软件包不但简化了这种应用，而且符合 L^ATeX 语法。

这个软件包可以像通常那样在导言中用下面命令调用：

```
2.8 \usepackage{ifthen}
```

或者把它指定为 `\documentstyle` 的一个选项, 即

```
2.09 \documentstyle[...ifthen,...]{...}
```

这样就可以使用 `\ifthenelse` 和 `\whiledo` 这两条命令了, 它们的语法是:

```
\ifthenelse{ 测试条件 }{then 文本 }{else 文本 }
\whiledo{ 测试条件 }{do 文本 }
```

在这两种语法中, 测试条件 是一个逻辑声明(下面解释); 在第一条命令中, 如果这个声明等于 *<true>*, 那么就把 *then* 文本 插入到正文中, 否则就把 *else* 文本 插入到正文中。在第二条命令中, 只要 测试条件 等于 *<true>*, *do* 文本 被插入(执行)。(*do* 文本 必须改变对 测试条件 的输入, 否则它永不会停止!) 文本中也可以包含命令, 甚至可以定义或重定义命令。

共有四种类型的逻辑声明, 可以把它们组合起来, 以形成复杂的声明。

测试数字

要比较两个数字, 或者值为数字的命令, 只要在它们中间放上关系运算符 *<*, *=* 或 *>* 就可以了, 这三个符号分别代表小于, 等于或大于。计数器中的值可以通过把它的名称作为 `\value` 命令的参数值来进行测试。例:

```
\newcommand{\three}{3}
\ifthenelse {\three = 3} {O.K.} {What?}
\ifthenelse {\value{page} < 100 }
{Page xx} {Page xxx}
```

在第一种情形中是显示出 ‘O.K.’, 因为 `\three` 等于 3; 在第二种情形中, 如果当前页码小于 100, 就会显示出 Page xx, 否则就显示出 Page xxx。(上面加的空格是为了明了, 因为参数值中间的空格总是被忽略。)

要测试一个数是偶数, 还是奇数, 可以用 `\isodd`2.8 命令:

```
\ifthenelse {\isodd{\value{page}}}
{odd} {even}
```

测试文本

若要测试两条命令得到的是否是相同一块文本, 或者一条命令是否定义成特定的字符串, 可以用

```
\equal{ 字符串一 }{ 字符串二 }
```

这里的 字符串一 和 字符串二 就是文本或者可以简化为文本的命令。例如, 给出如下定义后:

```
\ifthenelse {\equal{\name}{Fred}} {Fredrick} {??}
```

若 `\name` 已被定义成 Fred(用命令 `\newcommand`), 那么就会插入 Fredrick, 否则就显示出两个问号。

测试长度

另一个逻辑声明用来比较两个长度。

$\boxed{2\varepsilon}$ `\lengthtest{关系}`

这里的 关系 由中间用关系运算符 <, = 或 > 分开的两个长度或长度命令组成。例如,

```
\newlength{\horiz} \newlength{\vert}
\newlength{\min}
. . . . .
\ifthenelse {\lengthtest{\horiz > \vert}}
  {\setlength{\min}{\vert}} {\setlength{\min}{\horiz}}
```

会把 \min 取成 \horiz 和 \vert 两者中较小的那个。

测试开关

所谓 Boolean 开关就是要么为 *<true>*, 要么为 *<false>* 的参数, 也称为标志。存在处理标志的三条命令:

$\boxed{2\varepsilon}$ `\newboolean{字符串}` 创建一个新开关
 $\boxed{2\varepsilon}$ `\setboolean{字符串}{数}` 赋值 true 或 false
 $\boxed{2\varepsilon}$ `\boolean{字符串}` 测试其值

其中最后那个就可以用于 `\ifthenelse` 和 `\whiledo` 命令中的 测试条件。

在 L^AT_EX 中还有很多内部开关, 可以测试它们的值 (但从不要重设它们的值!)。其中最有用的就是 `@twoside` 和 `@twocolumn`, 可以用来测试当前激活的是不是双面模式或者两列模式。由于其中包含字符 @, 因此它只能用在类或宏包文件中 (附录 C)。

组合逻辑声明

可以把上面列出的逻辑声明利用下面的逻辑运算符, 组合成更复杂的声明:

`\and` `\or` `\not` `\(` `\)`

只要熟悉 Boolean 逻辑, 这些运算符的含义是非常明了的。例如, 如果激活了两列模式或者 `\paperwidth` 大于 15cm 且同时页码计数器的值小于 100 时, 就设置 `\textwidth` 等于 10cm, 那么下面的定义:

```
\ifthenelse {\lengthtest{\textwidth > 10cm} \or
  \(\lengthtest{\paperwidth > 15cm} \and
    \vaule{page} < 100 \) }
  {\setlength{\textwidth}{10cm}} {}
```

就可以达到这个目的。

上面这个条件比较复杂，就非常适用于定义可以有不同行为的新命令，或者包含在宏包和类文件中（附录 C）。

下面给出一个相对简单的例子：假设作者不知道出版者用的到底是英国拼写，还是美国拼写。那么他可以在文档中把两种拼写都包含进去，在导言中加进一个开关，以得到最后的选择。

```
\newboolean{US}
\setboolean{US}{true} %For American spelling
%\setboolean{US}{false} %For British spelling
\newcommand{\spell}[2]{\ifthenelse{\boolean{US}}{#1}{#2}}
```

那么现在 `\spell` 命令就会显示出根据标志 `US` 的设置来确定显示第一个还是第二个参数值。因此在正文中作者可以如下输入：

```
... the \spell{color}{colour} of music ...
```

这样如果指定的是 `\setboolean{US}{true}`，就会生成美国拼法，否则得到的就是英国拼法。事实上，不同工作部分之间可以用不同的拼法，只需简单地在适当地方改变开关的值就可以了。（在写书时这就是一种好方法，因为编辑在最终决定手稿之前是有可能改变拼法的。）

`\whiledo` 的一个例子：作者希望把某一文本重复写 n 遍，这里的文本和 n 是可变的，那么可以用如下方法：

```
\newcounter{mycount}
\newcommand{\replicate}[2]{\setcounter{mycount}{#1}
\whiledo{\value{mycount}>0}{#2\addtocounter{mycount}{-1}}}
```

那么 `\replicate{30}{?}` 就会显示出 30 个问号。

§7.4 用户定义的环境

可以用下面的命令来创建或修改环境：

```
2ε \newenvironment { 环境名 } [ 参数个数 ] [ 可省参数 ]
{ 开始的定义 } { 结束的定义 }
2ε \renewenvironment { 环境名 } [ 参数个数 ] [ 可省参数 ]
{ 开始的定义 } { 结束的定义 }
```

或者

```
2.09 \newenvironment { 环境名 } [ 参数个数 ]
{ 开始的定义 } { 结束的定义 }
2.09 2ε \renewenvironment { 环境名 } [ 参数个数 ]
{ 开始的定义 } { 结束的定义 }
```

这里参数值意义如下：

环境名: 对于 `\newenvironment` , 它不可以与任一已存在的 \LaTeX 或用户定义的环境或命令重名。另一方面, 对 `\renewenvironment` , 则就必须有同名的环境存在。要对 \LaTeX 环境进行任何修改, 都要求用户对 \LaTeX 的内部工作机理有相当深入的了解。

参数个数: 介于 1 到 9 之间的数, 说明环境有多少个参数值; 如果忽略了这个可省参数值, 环境就没有参数值。

可省参数: 如果第一个参数值 (`#1`) 是可省的, 这是它的默认值; 它与 `\(re)newcommand` (166 页) 中的同名参数值行为一样。

开始的定义: 当 `\begin{环境名}` 被调用时, 被插入的初始化文本; 如果这一文本中包含形如 `#n` , $n = 1, \dots$, 参数个数 项, 那么环境就要用如下形式调用:

`\begin{环境名}{参数 1}...{参数 n}...`

在开始的定义中出现的每个 `#n` 都要用 参数 n 取代。

结束的定义: 当调用了 `\end{环境名}` 时, 要插入的结束文本; 这里不要用哑参数值 `#n` , 因为它们只允许出现在开始的定义文本中。

§7.4.1 没有参数的环境

同用户定义命令中的情形一样, 这里首先讲的环境也是没有可省参数值参数个数的。用户若要定义一个自己的环境 `sitquote` , 可以如下输入:

```
\newenvironment{sitquote}{\begin{quote}\small
\itshape}{\end{quote}}

which sets the text appearing between \begin{sitquote} text \end{sitquote}
in the typeface \small\itshape, and indented on both sides from the main
margins, as demonstrated here.
```

在这里开始的定义由命令序列 `\begin{quote}\small\itshape` 组成, 而结束的定义只有 `\end{quote}` 。那么现在调用

```
\begin{sitquote} 文本 \end{sitquote} 就等价于
\begin{quote}\small\itshape 文本 \end{quote}
```

这样就得到了所期望的结果。

这个例子不是很好, 因为只要在 `quote` 环境的开始加上 `\small\itshape` 就很容易得到同样的效果。然而, 如果在文档中经常出现这样的结构, 那么这个新环境对于简化输入和减少在输入 `\small\itshape` 时出错的机会方面就非常有效了。

我们现在把前面这个例子做如下的推广:

```
\newcounter{com}
\newenvironment{comment}
{\noindent\slshape Comment:\begin{quote}\small\itshape}
```

```
{\stepcounter{com}\hfill(\arabic{com})\end{quote}}
```

这里 开始的定义 就是由文本和命令组成:

```
\noindent\slshape Comment:\begin{quote}\small\itshape
```

而 结束的 定义 组成为:

```
\stepcounter{com}\hfill}\arabic{com})\end{quote}
```

其中 `com` 是一个由 `\newcounter` 创建的用户计数器。由于 `\begin{comment}` 命令要在环境开始处插入 开始的定义 文本, `\end{comment}` 在结束处插入 结束的 定义 文本, 因此很容易知道:

```
\begin{comment} This is a commen.
```

```
Comments should ...
```

```
... in round parenthesis.
```

```
\end{comment}
```

的结果应是如下结构:

Comment:

This is a comment. Comments should be preceded by the word Comment: the text being in a small, italic typeface, indented on both sides from the main margins. Each comment receives a running comment number at the lower right in round parentheses. (1)

读者应仔细对比一下环境定义中的命令序列与例子中文本位置的关系, 这样可以准确地知道这个环境的效果。这里有两个非常明显的缺陷, 那就是当调用 `\begin{comment}` 时其正好位于一行的中间, 前面没有空行的情形, 或者注释的最后一行太长, 使得该行没有足够的空间把活动编号放在行尾时的情形。

下面这个修正版本就解决了这两个问题:

```
\renewenvironment{comment}
```

```
{\begin{sloppypar}\noindent\slshape Comment:
```

```
\begin{quote}\small\itshape}
```

```
{\stepcounter{com}\hspace*{\fill}\arabic{com})\end{quote}
```

```
\end{sloppypar}}
```

这里利用 `\begin{sloppypar}` 命令使得环境总是开始一个新段, 而且在断行是不会有满行情形发生。如果注释的编号在最后一行放不下来, 就会开始一个新行, 编号是右对齐的, 因为这里用了命令 `\hspace*{\fill}`。这里同样希望读者仔细查看插入在 `\begin{comment}` 和 `\end{comment}` 之间的定义文本。

§7.4.2 有参数的环境

向环境中传递参数值, 同命令中的情形是完全一样的。举例来说, 我们对

上面的注释环境进行修正,使得注释人的姓名加在单词 `Comment:` 的后面;当环境被激活时这个姓名就是其参数值。

```
\renewenvironment{comment}[1]
{\begin{sloppypar}\noindent\slshape Comment: #1
\begin{quote}\small\itshape}
{\stepcounter{com}\hspace*{\fill}\(\arabic{com})}%
\end{quote}\end{sloppypar}}

那么现在输入

\begin{comment}{Helmut Kopka} This is a modified ...
... environment argument \end{comment}
```

结果就是:

Comment: Helmut Kopka

This is a modified comment. Comments should be preceded by the word Comment:, followed by the name of the commenter, with the text of the comment being in a small, italic typeface, indented on both sides from the main margins. Each comment receives a running comment number at the lower right in round parentheses. The name of the commenter is transferred as an environment argument. (2)

下面再对这个例子进行修正,交换注释编号与注释人姓名的位置。把活动编号放在单词 `Comment:` 后面是没有任何问题的,因为这只需要简单地把来自于 `{\结束的定义}` 命令插入在上面 `#1` 哑元参数值的地方。然而,要把 `#1` 符号放在原来注释号所处的地方,在 \LaTeX 处理过程中就会出现错误信息,因为这与 `\newenvironment` 命令的语法相冲突: ‘在 `{\结束的定义}` 中不能有哑元参数值’。如果哑元参数值位于 `\begin{quote}` 的后面,那么其位置就不是所期望的地方,而是处于注释文本的开始。

下面给出解决这个问题的技巧:

```
\newsavebox{\comname}
\renewenvironment{comment}[1]
{\begin{sloppypar}\noindent\stepcounter{com}\slshape
Comment \arabic{com}\sbox{\comname}{#1}
\begin{quote}\small\itshape}
{\hspace*{\fill}\usebox{\comname}\end{quote}\end{sloppypar}}
```

我们在 4.7.1 节中已讲述了 `\newsavebox`, `\sbox` 和 `\usebox` 命令。这里 `\comname` 就是用 `\newsavebox` 命令提前创建的盒子名称,其中放的是第一个参数值,即注释人的姓名。利用这个新的定义,注释的新样子如下:

Comment 3

In this form, every comment is assigned a sequential number after the word

Comment. The comment text appears as before, while the name of the commenter is entered as the environment argument and is placed at the right of the last line. *Helmut Kopka*

要实现不只一个参数值的环境，步骤同这里是完全一样的，因此这里不再讲述。

§7.4.3 有一个可省参数值的环境 2ε

同命令一样，环境也可以有一个可省参数值。还是考虑上面那个例子，如果我们觉得绝大多数注释都是由 Helmut Kopka 给出的，这样我们就可以把定义的第一行改为

```
\renewenvironment{comment}[1][Helmut Kopka]{...}{...}
```

因此只有当注释人是其它人时才需要指定。

```
\begin{comment}[Patrick W. Daly] More than ...
... appropriate. \end{comment}
```

结果为

Comment 4

More than one person may want to make a comment, but perhaps one person makes more than others do. An optional argument for the name is then appropriate *Patrick W. Daly*

练习 7.8: 推广注释环境的定义，使得不能在标题 ‘Comment n’ 与注释正文之间以及正文与注释人姓名之间有分页。

提示：解法应该利用 `samepage` 环境或者 `\nopagebreak` 命令。

练习 7.9: 利用 `minipage` 环境创建一个新的环境，名称为 `varbox`，它具有一个参数值，该参数值是一个文本样本，小页环境的宽度由它确定。因此

```
\begin{varbox}{‘As wide as this sample text’}
. . . . . \end{varbox}
```

就会用一个宽度等于文本 ‘As wide as this sample text’ 的自然宽度的小页环境包围其中文本。

提示：首先要建立一个用户定义的长度参数，名称可以是 `\verwidth`。在 7.2 节中有关于文本宽度怎样赋给长度参数的讲解。

练习 7.10: 生成一个名称为 `varlist` 的环境，它有两个参数值，行为类似于 4.4.3 节的示例列表的推广。第一个参数值就是每次调用 `\item` 时显示出来的项词；第二个参数值项枚举的编号样式。例如，利用

```
\begin{varlist}{Sample}{\Alph}. . . . \end{varlist}
```

在环境中依次调用 `\item` 时，生成序列 ‘Sample A’, ‘Sample B’, ...。缩进应比项词的宽度大 1cm，而项词本身是在标签盒子中左对齐。

提示：解法中还是要借助于用户定义长度，如 `\itemwidth`。当项词的长度利用 `\settowidth` 保存起来后，缩进可以如下设置：

```
\setlength{\leftmargin}{\itemwidth}
```

这样缩进就等于项词长度，然后

```
\addtolength{\leftmargin}{1cm}
```

就可以使它增大 1cm。对于 `\labelwidth` 和 `\labelsep` 的长度赋值也可以类似进行。关于其它的细节可以参考 4.4 节。

§7.5 对用户定义结构的解释

本节包含了用户定义 \LaTeX 结构的创建和使用方面的一些一般性注解。这没有严格的规则，只是阐述了我们根据自己的经验而得到的一些想法。每个用户都会根据自己的经验，总结出自己的实用方法。

§7.5.1 保存用户定义的结构

用户创建自己定义的结构，可以简化大量的工作。经常使用的命令和 / 或文本应利用 `\newsavebox`、`\newcommand` 或 `\newenvironment` 定义单独写到一个文件中。这个文件可以用 `\input` 命令读入，从而在其它文档文件中也可以使用。

随着时间的推移，这样就会积累出大量的结构。若把它们全部放到一个文件中就会减慢处理过程，而且跟踪内容和命令名称也变得非常困难。因此我们推荐用户结构应根据应用领域保存在许多独立的文件中。

§7.5.2 缩写结构

用户定义命令的一个简单应用就是缩短 \LaTeX 结构的名称。例如，利用

```
\newcommand{\be}{\begin{enumerate}}
```

```
\newcommand{\ee}{\end{enumerate}}
```

只要输入 `\be` 就可以开始一个 \LaTeX 环境 `enumerate`，而 `\ee` 用来结束这个环境。

这样的缩写可以显著地避免大量输入，但它并不适合于做为命令大量保存起来，因为容易忘记命令名称背后所包含的意义。 \LaTeX 的作者 Leslie Lamport 已经精心选择了命令或环境的名称，以明白地表示它们的功能。这种明确完整的命名，要比不明了的任何缩写容易记住。但是可以在单个文档内容中使用某些缩写，只要不太多就可以。具体要视用户的输入能力来确定了。

§7.5.3 命令和记数器的名称相同

在前面的例子中，一些特定的命令或环境的应用中使用了许多记数器，例如 171 页上 `\myfootnote` 命令中的 `myfn`，177 页上 `comment` 环境中的

com。在这些情形中，计数器与对应的命令或环境有不同的名称。但这并不是一定要这样：计数器可以具有与命令或环境相同的名称。*L^AT_EX*可以从上下文知道某一名称指的是计数器还是命令 / 环境。

在上面例子中之所以选择不同的名称，就是为了避免初学者混淆。事实上完全有理由把计数器的名称同其所对应的命令或环境的名称取成一样的，*L^AT_EX*本身就是经常这样做的（7.1.1 节）。在上面所讲到的例子中，计数器就可以命名为 `myfootnote` 和 `comment`，以强调它与同名命令和环境之间的依赖关系。

§7.5.4 用户定义的作用范围

在导言中给出的用户结构对整篇文档都有效。在环境内给出的命令和环境定义有效性持续到包围环境的结束。甚至在定义所处的环境外面 *L^AT_EX* 连它们的名称也不知道。因此如果在另一个环境又要进行同名的定义，应该用 `\newcommand` 或 `\newenvironment`，而不是 `\renew` 形式的命令。

对于全局定义的命令和环境（即放在导言中），对它们进行新的定义需要用 `\renew` 形式的命令。然而这个定义只在其所处的环境内有效；在这个环境外面，原先的全局定义仍然有效。

这一规则同样适用于嵌套环境中的结构定义。在外部环境定义在内部环境中有效，在深层必须有 `\renewcommand` 或 `\renewenvironment` 进行同名新的定义。当离开内层环境后，新定义不再有效，而重新恢复原先的含义。

警告：用 `\newsavebox` 或 `\newcounter` 创建的结构是全局定义的。如果在环境中给出，即使在环境外面它也是有效的。同样虽然 `\savebox` 并不是全局性的，但 `\setcounter` 却是全局性的。

§7.5.5 定义的顺序

用户定义的结构可以彼此嵌套。如果一个用户定义结构中包含另一个用户定义结构，通常内部的结构已经被定义了。然而，这并不是必需的。用户定义中可以包含后面定义的其它用户结构。重要的是当第一次调用命令之前其它结构已经有了定义。

例如，下面是一个正常的定义顺序：

```
\newcommand{\A}{定义 A}
\newcommand{\B}{定义 B}
\newcommand{\C}{\A \B}
```

这里 `\C` 是用 `\A` 和 `\B` 定义的。然而，也可以像下面这样输入：

```
\newcommand{\A \B}
没有调用 \C 的普通文本
\newcommand{\B}{定义 A}
```

```
\newcommand{\A}{定义 B}
```

其它文本, 可以随意调用 \A, \B 和 \C.

§7.5.6 传递参数值

在命令和环境定义中的哑元参数值可以用作定义中其它命令的参数值。

例如, 如果命令 \A 和 \B 每个都有一个参数值, 命令定义

```
\newcommand{\C}[3]{\A{#1}#2\B{#3}}
```

是可以接受的。这里 \C 的第一个和第三个参数值被传递给命令 \A 和 \B, 只有第二个参数值直接进入定义。可以把直接参数值和传递参数值进行任意的组合。下面是一个比较具体的例子:

```
\newcommand{\sumvec}[4]{\anvec{#3}{#4} = #1_1+#2_1,\ldots,
#1_#4+#2_#4}
```

这样调用 `\sumvec xyzn` 就得到 $z_1, \dots, z_n = x_1 + y_1, \dots, x_n + y_n$, 这里 \anvec 就是在 7.3.2 节中定义的命令。

§7.5.7 嵌套定义

用户定义可以彼此嵌套。类似于下面这样的结构

```
\newcommand{\外部命令名}{\newcommand{\内部命令名}{...}}
```

是可以的。根据 182 页上关于定义作用范围的说明, 用名称 {\内部命令名} 定义的命令只在命令 {\外部命令名} 内部才被知道和有效。虽然 TeX 的宏利用了大量的嵌套命令定义, 以限制临时性命令的寿命, 但是我们并不推荐过分地使用 LaTeX 定义嵌套, 因为这样太容易造成括号不匹配。忘记一个括号匹配, 将会在第二次调用外部命令时给出错误信息, 因为这时还残留第一次调用时一部分内部定义。但是我们还是给出下面这个例子:

```
\newcommand{\twentylove}
{
  {\newcommand{\fivelove}
    {
      {\newcommand{\onelove}
        {I love \LaTeX!}%
        \onelove\ \onelove\ \onelove\ \onelove\ \onelove}}}
    \fivelove\ \fivelove\ \fivelove\ \fivelove\ \fivelove}}
```

那么现在 `My opinion of \LaTeX:\ \twentylove` 结果为:

My opinion of LaTeX:

I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX!

I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX!

I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX!

I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX! I love LaTeX!

在定义中像上面那样把行缩进，可以帮助跟踪嵌套层次；同一层次中的每一行不考虑括号的前提下开始于同一列。

如果内部和外部定义中都有参数值，那么内外表示哑元参数值的符号必须不一样。内层定义的符号是 `##1 ... ##9`，而外层定义的符号 `#1 ... #9`。例如，

```
\newcommand{\thing}[1]{\newcommand{\color}[2]{The ##1 is ##2.}
\color{#1}{red} \color{#1}{green} \color{#1}{blue}}}
```

那么 `The colors of the objects are\`

```
\thing{dress}\ \thing{book}\ \thing{car}
```

 结果为

The colors of the objects are

The dress is red. The dress is green. The dress is blue.

The book is red. The book is green. The book is blue.

The car is red. The car is green. The car is blue.

像 7.5.5 节中那样分开定义和调用是比较容易接受的方法：

```
\newcommand{\thing[1]}{\color{#1}{red} \color{#1}{green}
\color{#1}{blue}}
\newcommand{\color}[2]{The #1 is #2.}
```

§7.5.8 不期望的空格

有时候用户定义结构中会出现不期望的空格，或者出现比需要多的空格。这几乎总是由定义中空白或新行造成的，在定义中包含空格只是为了提高源文本可读性，当结构被调用时它们可能被解释成空白。

例如，如果在 171 页上的 `\myfootnote` 定义中去掉第一行的 `%` 字符，那么就会在此处加入一个分行符，从而转化成一个空白。这个空白要插入在接受脚注标记的前面单词与调用 `\footnote` 命令实际生成标记之间，结果就会使得标记同单词分开。

在此处我们想提醒读者许多 *LaTeX* 命令是看不到的，即在调用它们的地方不会生成任何文本。如果在这样不可见命令与周围命令之间有空格，那么就可能出现两个空格。

```
For example \rule{0pt}{0pt} produces
```

结果为 ‘For example produces’，有一个两倍单词间距。没有参数值的不可见命令不会导致这种问题，因为跟在命令后面的空格被当做终止符，从而消失了。而且，下面的 *LaTeX* 命令和环境总是去掉后续空格，即使它们有参数值：

```
\pagebreak \linebreak \label \glossary \vspace figure
\nopagebreak \nolinebreak \index \marginpar table
```

§7.5.9 最后两个例子

通常的 `description` 环境（4.3.3 节）把标签设成黑体，所有接在第一行

后面的行都缩进一个固定长度。为了描述计算机代码中的命令名，可能更希望用打字机字体（`\texttt` 命令或者 `\ttfamily` 声明）显示标签，后面的行缩进量等于最长标签的长度。为此我们现在定义一个新环境 `ttscript`，其使用方法为

```
\begin{ttscript}{ 取样文本 } 列表文本 \end{ttscript}
```

参数值 取样文本 在使用 `\texttt` 命令时的宽度等于左边缩进量。列表文本由 `\item[标签文本]` 命令后接相应的解释性文本组成，其中每行相对于左边界缩进 取样文本 的宽度。标签文本 在标签盒子中是用 `\texttt` 左对齐显示的。`ttscript` 环境的定义为

```
\newenvironment{ttscript}[1]{%
  \begin{list}{}{%
    \settowidth{\labelwidth}{\texttt{#1}}
    \setlength{\leftmargin}{\labelwidth}
    \addtolength{\leftmargin}{\labelsep}
    \setlength{\parsep}{0.5ex plus0.2ex minus0.2ex}
    \setlength{\itemsep}{0.3ex}
    \renewcommand{\makelabel}[1]{\texttt{##1\hfill}}}
  \end{list}}
```

这也很容易把标签的字体改成其它字体样式，重新给这个结构起个名字，把它保存起来以供一般性应用。

对上面定义中的前七行的理解应该没有任何问题。环境的定义包含一个参数值，它被如下传递：

```
\settowidth{\labelwidth}{\texttt{#1}}
```

这样就设置了标签的宽度。接着左边界的设置为：

```
\setlength{\leftmargin}{\labelwidth}
```

这样其值与标签 `\labelwidth` 是一样的，最后

```
\addtolength{\leftmargin}{\labelsep}
```

在左页边界中增加了标签分隔 `\labelsep` 的量。

第六行和第七行把 `\parsep` 和 `\itemsep` 设置成适当的值，用户可以按需要进行改变。如果必要的话，也可以包含其它的列表参数。

最后一行重定义了 `\makelabel`，这里需要加以解释。在 4.4.1 节中简要介绍了这一命令。它只在 `list` 环境中才有效，每当调用 `\item` 时就用它生成实际的标签。它通常的参数值就是 `\item` 命令的可省参数值。利用上面的重定义，现在的标签是包括字体命令 `\texttt` 后接 `\hfill`，这样在标签盒子中是左对齐的。由于重定义是嵌套在 `ttscript` 环境定义中，因此它的哑元参数值应是 `##1`，而不是 `#1`，在上一节中有解释。

例如，给出如下文本：

```

\begin{ttscript}{description}
\item[list] environments are useful for . . .
\item[enumerate] environments number . . .
\item[itemize] environments mark the . . .
\item[description] environments label . . .
\end{ttscript}

```

那么我们可以得到如下关于 *LT_EX* 列表环境的描述:

```

list      environments are useful for generating lists in which the various
           items are set off from one another for greater clarity;

enumerate environments number their items consecutively, starting at 1;

itemize   environments mark the various items with a distinguishing sym-
           bol, often a bullet •;

description environments label the items with some text in bold face type,
           for greater stress.

```

练习 7.11: 把 *ttscript* 环境推广成更一般的 *varscript* 环境, 使它具有两个参数值, 第二个参数值确定标签的字体样式。

用户定义的命令和环境可以有多达 9 个参数值。一个结构拥有的参数值越多, 其弹性就越大。另一方面, 调用命令时就变得很复杂和冗长, 因为参数的数目和顺序都必须严格与定义一致。

在 4.4.4 节中 68 页上的用户定义示例环境 *figlist* 中没有参数值。当调用它时, 每条 *\item* 命令生成 **Figure 1:**, **Figure 2:** 等等标签。而且文本相对于包围文本左边界缩进 2.6cm, 相对于右边界缩进 2cm。在列表声明中的类似于 *\labelsep*, *\parsep* 和 *\itemsep* 等其它列表参数取的是固定值。然而, 进行了下述定义后,

```

\newcounter{itemnum} \newlength{\addnum}
\newenvironment{genlist}[8]
{
  \begin{list}{\textbf{#1 \arabic{itemnum}:}}
  {
    \usecounter{itemnum}
    \settowidth{\labelwidth}{\textbf{#1}}
    \settowidth{\addnum}{\textbf{\arabic{itemnum}: }}
    \addtolength{\labelwidth}{\addnum}
    \setlength{\labelsep}{#2}
    \setlength{\leftmargin}{\labelwidth}
    \addtolength{\leftmargin}{\labelsep}
    \setlength{\rightmargin}{#3}
    \setlength{\listparindent}{#4}
  }
}

```



```

\setlength{\parsep}{#5}
\setlength{\itemsep}{#6}
\setlength{\topsep}{#7#8}}
{\end{list}}

```

就能生成一个广义列表,其中第一个参数确定每次调用 `\item` 命令时与顺序编号显示在一起的共同项名称。左边的缩进设成项名称宽度加上 `\labelsep`,后者由第二个参数值给定。后面五个参数值确定各种列表参数值的长度。最后一个参数值为指定环境中文本的字体的声明。这个新环境的语法为:

```

\begin{genlist}{项名称}{标签间距}{右边距}{列表段落缩进}
  {段落间距}{项间距}{顶部间距}{字体样式}
\item 文本
...
\item 文本
\end{genlist}

```

当如下调用

```

\begin{genlist}{Sample}{4mm}{1cm}{0pt}{1ex plus0.5ex}
  {0pt}{0pt}{\slshape}
\item no value
\item The enclosed coupon is worth a value of \$2.00 towards
  the purchase of your next order.
\end{genlist}

```

结果为:

Sample 1: *no value*

Sample 2: *The enclosed coupon is worth a value of \$2.00 towards the purchase of your next order.*

在这个例子中,每个长度项都必须有单位。当然也可以在定义中直接包含单位,这样在调用时只需给出数值就可以了。同样竖直距离中的橡皮长度也可以用如下定义中的算法给出:

```

\setlength{\parsep}{#5ex plus0.3#5ex minus0.5#5ex}

```

这里要求第五个参数值为纯粹的数字。如果其值为 2, `\parsep` 就会等于 `2ex plus0.6ex minus1ex`。这里还要提出的是,一定在要简化项数与记住其含义的复杂之间取得折衷,特别是如果各种不同长度参数值具有不同的大小和算法的时候更要如此。