7 Programmability

We've touched several times on the ability of LaTeX to be reprogrammed. This is one of its central features, and one that still, after nearly a quarter of a century, puts it well above many other typesetting systems, even those with programming systems of their own. It's also the one that needs most foreknowledge, which is why this chapter is in this position.

TeX is basically a programming language for typesetting. As such, it allows you to define ***macros***, which are little (or large) program-like sequences of commands with a name which can be used as a command itself. This in effect makes a macro a shorthand for a sequence of operation you wish to perform more than once. LaTeX is in fact just a large collection of such macros.

Macros can be arbitrarily complex. Many of the ones used in the standard LaTeX packages are several pages long, but as we will see, even short one-liners can very simply automate otherwise tedious chores and allow the author to concentrate on the most important thing; *writing*.

7.1 Simple replacement macros

In its simplest form, a LaTeX macro can just be a straightforward text replacement of a phrase to avoid lengthy retyping with the possibility of misspelling something each time you need it, eg

```
\newcommand{\EF}{European Foundation for the
    Improvement of Living and Working Conditions}
```

Put this in your Preamble, and you can then use `\EF` in your document and it will typeset it as the full text. Remember that after a command ending in a letter you need to leave a space to avoid the next word getting gobbled up as part of the command (see section 1.5.1 on page 13). If you want to force a space to be printed after the expansion, use a backslash followed by a space, eg

```
The \EF\ is a member institution of the Commission of the
 European Union.
```

As you can see from this example, the `\newcommand` command takes two arguments: the name you want to give the new command, and the expansion to be performed when you use it, so there are always two sets of curly braces after a `\newcommand`. The names of new commands created like this MUST be made of the letters A–Z and a–z *only*, and must not be the names of existing commands.

7.2 Macros using information gathered previously

A more complex example is the macro `\maketitle` which is used in almost every LaTeX document to format the title block. In the default document classes (book, report, and article) it performs small variations on the layout of a centred block with the title followed by the author followed by the date, as we saw in section 2.3 on page 42.

If you inspect one of the default document class files, such as `report.cls` you will see `\maketitle` defined (and several variants called `\@maketitle` for use in different circumstances). It uses the values for the title, author, and date which are assumed already to have been stored in the internal macros `\@title`, `\@author`, and `\@date` by the author using the matching `\title`, `\author`, and `\date` commands in the document *before* using the `\maketitle` command.

This use of one command (eg `\title`) to store the information in another (eg `\@title`) is a common way of gathering the information from the user. The use of macros containing the `@` character prevents

their accidental misuse by the user because they designed for use in packages and classes, and are disallowed in document text. To use them in your Preamble (eg to redefine `\maketitle`), you have to allow the `@` sign to temporarily become a 'letter' using the `\makeatletter` command so the `@` can be recognised in a command name (and remember to turn it off again afterwards with the `\makeatother` command — see item 1 on page 163 below).

```
\makeatletter
\renewcommand{\maketitle}{%
   \begin{flushleft}%
      \sffamily
      {\Large\bfseries\color{red}\@title\par}%
      \medskip
      {\large\color{blue}\@author\par}%
      \medskip
      {\itshape\color{green}\@date\par}%
      \bigskip\hrule\vspace*{2pc}%
   \end{flushleft}%
}
\makeatother
```

Insert this immediately before the `\begin{document}` in the sample file in the first listing, and make sure you have used the xcolor package. Typeset the file and you should get something like Figure 7.1 on page 164.

In this redefinition of `\maketitle`, we've done the following:

1. Enclosed the changes in `\makeatletter` and `\makeatother` to allow us to use the `@` sign in command names;[1]
2. Used `\renewcommand` and put `\maketitle` in the first pair of curly braces after it;
3. Opened a second pair of curly braces to hold the new definition. The closing curly brace of this pair is immediately before the `\makeatother`;

[1] If you move all this Preamble into a package (`.sty`) file of your own, you don't need these commands: the use of `@` signs in command names is allowed in package and class files.

4. Inserted a `flushleft` environment so the whole title block is left-aligned;

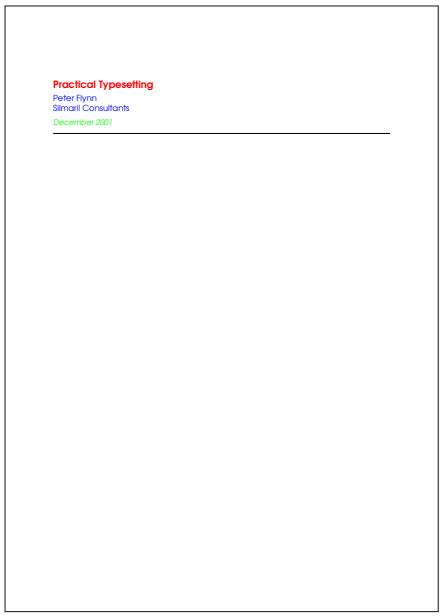
5. Used `\sffamily` so the whole title block is in the defined sans-serif typeface;

6. For each of `\@title`, `\@author`, and `\@date`, we have used some font variation and colour, and enclosed each one in curly braces to restrict the changes just to each command. The closing `\par` of each one makes sure that multiline title and authors and dates would get typeset with the relevant line-spacing;

7. Added some flexible space between the lines, and around the `\hrule` (horizontal rule) at the end;

Note the `%` signs after any line ending in a curly brace, to make sure no intrusive white-space find its way into the output. These aren't needed after simple commands where there is no curly brace because excess white-space gets gobbled up there anyway.

Figure 7.1: Example of reprogrammed title layout



Exercise 16. Rewriting the title

Add the code above to your test document (or create a new one), then add title, author, and date, and make your new title.

7.3 Macros with arguments

But macros are not limited to text expansion or the reproduction of previously-stored values. They can take arguments of their own, so you can define a command to do something with specific text you give it. This makes them much more powerful and generic, as you can write a macro to do something a certain way, and then use it hundreds of times with a different value each time.

We mentioned earlier (in section 6.2.8 on page 154) the idea of making new commands to put specific classes of words into certain fonts, such as `\foreign` or `\product`. Here's an example for a new command `\tmproduct`, which also indexes the product name and adds a trademark sign:

```
\newcommand{\tmproduct}[1]{%
        \textit{#1}\texttrademark%
        \index{#1@\textit{#1}}%
}
```

If I now type `\tmproduct{Velcro}`, I get *Velcro*™ typeset, and if you look in the index, you'll find this page referenced under *Velcro*. Let's examine what this does:

1. The macro is specified as having one argument (that's the `[1]` in the definition). This will be the product name you type in curly braces when you use `\product`. Macros can have up to nine arguments.

2. The expansion of the macro is contained in the second set of curly braces, spread over several lines (see item 5 on page 166 for why).

3. It prints the value of the first argument (that's the `#1`) in italics, which is conventional for product names, and adds the `\texttrademark` command.

4. Finally, it creates an index entry using the same value (`#1`), making sure that it's italicised in the index (see the list item 'Font changes' section 5.4.1 on page 123 to remind yourself of how indexing something in a different font works).

5. Typing this macro over several lines makes it easier for humans to read. I could just as easily have typed

```
\newcommand{\product}[1]{\textit{#1}\index{#1@\textit{#1}}}
```

   but it wouldn't have been as clear what I was doing.

   One thing to notice is that to prevent unwanted spaces creeping into the output when LaTeX reads the macro, I ended each line with a comment character (`%`). LaTeX normally treats newlines as spaces when formatting (remember the list item '`%`' section 1.6.1 on page 16), so this stops the end of line being turned into an unwanted space when the macro is used. LaTeX usually ignores spaces at the *start* of macro lines anyway, so indenting lines for readability is fine.

In section 1.9.2 on page 26 we mentioned the problem of frequent use of unbreakable text leading to poor justification or to hyphenation problems. A solution is to make a macro which puts the argument into an `\mbox` with the appropriate font change, but precedes it all with a conditional `\linebreak` which will make it more attractive to TeX to start a new line.

```
\newcommand{\var}[1]{\linebreak[3]\mbox{\ttfamily#1}}
```

This only works effectively if you have a reasonably wide setting and paragraphs long enough for the differences in spacing elsewhere to get hidden. If you have to do this in narrow journal columns, you may have to adjust wording and spacing by hand occasionally.

7.4 Nested macros

Here's a slightly more complex example, where one macro calls another. It's common in normal text to refer to people by their forename and surname (in that order), for example Donald Knuth,

but to have them indexed as *surname, forename*. This pair of macros, \person and \reindex, automates that process to minimise typing and indexing.

```
\newcommand{\person}[1]{#1\reindex #1\sentinel}
\def\reindex #1 #2\sentinel{\index{#2, #1}}
```

1. The digit 1 in square brackets means that \person has one argument, so you put the whole name in a single set of curly braces, eg \person{Don Knuth}.

2. The first thing the macro does is output #1, which is the value of what you typed, just as it stands, so the whole name gets typeset exactly as you typed it.

3. But then it uses a special feature of Plain TEX macros (which use \def instead of LATEX's \newcommand[2]): they too can have multiple arguments but you can separate them with other characters (here a space) to form a pattern which TEX will recognise when reading the arguments.

   In this example (\reindex) it's expecting to see a string of characters (#1) followed by a space, followed by another string of characters (#2) followed by a dummy command (\sentinel). In effect this makes it a device for splitting a name into two halves on the space between them, so the two halves can be handled separately. The \reindex command can now read the two halves of the name separately.

4. The \person command invokes \reindex and follows it with the name you typed plus the dummy command \sentinel (which is just there to signal the end of the name). Because \reindex is expecting two arguments separated by a space and terminated by a \sentinel, it sees 'Don' and 'Knuth' as two separate arguments.

   It can therefore output them using \index in reverse order, which is exactly what we want.

[2] Don't try this at home alone, folks! This one is safe enough, but you should strictly avoid \def for a couple of years. Stick to \newcommand for now.

A book or report with a large number of personal names to print and index could make significant use of this to allow them to be typed as `\person{Leslie Lamport}` and printed as Leslie Lamport, but have them indexed as 'Lamport, Leslie' with virtually no effort on the author's part at all.

7.5 Macros and environments

As mentioned in section 4.6.3 on page 102, it is possible to define macros to capture text in an environment and reuse it afterwards. This avoids any features of the subsequent use affecting the formatting of the text.

One example of this uses the facilities of the fancybox package, which defines a variety of framed box commands to display your text, but they require a pre-formed box as their argument, so the package provides a special environment `Sbox` which 'captures' your text for use in these boxes.

Here we put the text in a `minipage` environment because we want to change the width; this occurs *inside* the `Sbox` environment so that it gets typeset into memory and stored in a box. It can then be 'released' afterwards with the command `\TheSbox` as the argument of the `\shadowbox` command (and in this example it has also been centred).

```
\begin{Sbox}
\begin{minipage}{3in}
This text is formatted to the specifications
of the minipage environment in which it
occurs.

Having been typeset, it is held in the Sbox
until it is needed, which is after the end
of the minipage, where you can (for example)
align it and put it in a special framed box.
\end{minipage}
\end{Sbox}
\begin{center}
\shadowbox{\TheSbox}
\end{center}
```

☞

This text is formatted to the specifications of the minipage environment in which it occurs.

Having been typeset, it is held in the Sbox until it is needed, which is after the end of the minipage, where you can (for example) centre it and put it in a special framed box.

The point about this kind of construct is that it can be turned into an environment of your own, so you can reuse it wherever you need:

```
\newenvironment{warning}{%
  \begin{Sbox}\begin{minipage}{8cm}}
  {\end{minipage}\end{Sbox}\begin{center}
   \shadowbox{\TheSbox}\end{center}}
```

7.6 Reprogramming LaTeX's internals

LaTeX's internal macros can also be reprogrammed or even rewritten entirely, although doing this can require a considerable degree of expertise. Simple changes, however, are easily done.

Recall that LaTeX's default document structure for the Report document class uses Chapters as the main unit of text, whereas in reality most reports are divided into Sections, not Chapters (footnote 6 on page 47). The result of this is that if you start off your report with `\section{Introduction}`, it will print as

0.1 Introduction

which is not at all what you want. The zero is the (missing) chapter number, because no chapter has been started. But this numbering is controlled by macros, and you can redefine them. In footnote 2 on page 78 we said that every counter automatically gets a related command beginning with 'the'. In this case what we need to change is that command `\thesection` because the way the counters are set up makes it reproduce the value of the counter *section* (see section 4.1.6 on page 79) *plus* any higher-level value (eg *chapter*). It's redefined afresh in each document class file, using the command `\renewcommand` (in this case in `report.cls`):

```
\renewcommand\thesection{\thechapter.\@arabic\c@section}
```

You can see it invokes `\thechapter` (which is defined elsewhere to reproduce the value of the *chapter* counter), and it then prints a dot, followed by the Arabic value of the counter called *section* (that `\c@` notation is LaTeX's internal way of referring to counters). You can redefine this in your Preamble to simply leave out the reference to chapters:

```
\renewcommand{\thesection}{\arabic{section}}
```

I've used the more formal modern method of enclosing the command being redefined in curly braces. For largely irrelevant historical reasons these braces are often omitted in LaTeX's internal code (as you may have noticed in the example earlier). And I've also used the 'public' version of the `\arabic` command to output the value of

section (LaTeX's internals use a 'private' set of control sequences containing @-signs, designed to protect them against being changed accidentally).

Now the introduction to your report will start with:

1 Introduction

What's important in making this type of modification is that you DO NOT alter the original document class file `report.cls` (ever): you just copy the command you need to change into your own document Preamble, or a private package file, and modify that instead. It will then override the default because it will get loaded *after* the document class.

7.6.1 Changing list item bullets

As mentioned earlier, here's how to redefine a bullet for an itemized list, with a slight tweak:

```
\usepackage{bbding}
...
\renewcommand{\labelitemi}{%
        \raisebox{-.25ex}{\PencilRight}}
```

Here we use the bbding package which has a large selection of 'dingbats' or little icons, and we change the label for top-level itemized lists (`\labelitemi`, find these in any document class file) to make it print a right-pointing pencil (the names for the icons are in the bbding package documentation: see section 3.1.3 on page 59 for how to get it).

In this case, we are also using the `\raisebox` command within the redefinition because it turns out that the symbols in this font are positioned slightly too high for the typeface we're using. The `\raisebox` command takes two arguments: the first is a dimension, how much to raise the object by (and a negative value means 'lower': there is no need for a separate `\lowerbox` command); and the second is the text you want to affect. Here, we are shifting the symbol down by 1/4ex (see section 1.9.1 on page 23 for a list of dimensional units LaTeX can use).

There are label item commands for each level of lists (1–4) which have command names ending in Roman numerals (i–iv) because of the rule that command names can only use letters. Thus to change the icon for the fourth-level list, modify `\labelitemiv`. There is a vast number of symbols available: see *The LaTeX Symbol List* for a comprehensive list.