Hour 13

User-Defined Macros

LaTeX provides many in-built commands and environments for preparing a document. Besides those, it permits to define new commands and environments. Moreover, in-built ones can also be redefined to alter their behaviors. Many such cases are already addressed in previous Hours, which are systematically discussed in this Hour. In this book, although the original LaTeX syntax are printed in **red colored** (for online version) **and boldfaced sans serif fonts** for their clear distinction, all user-defined syntax are printed in normal black color so that a reader is not misled.

13.1 Defining New Commands

It is a tedious job to insert a long command or a combination of commands, or even a piece of texts, if it is to be used repeatedly in a document. In that case, some short commands can be defined to represent such long items conveniently[1]. A new command is defined in the preamble through the **\newcommand{**`newc`**}{**`aval`**}** or **\providecommand{**`newc`**}{**`aval`**}** command, where `newc` is the new command to be defined and `aval` is the attribute to be represented by `newc`. In general, the name of a new command should be alphabetic only, not to start with '`end`' and not to match with an existing command. In the case of **\newcommand{}{}**, an error message will be generated if a new command matches with an existing command, while **\providecommand{}{}** will retain the existing command without any message, i.e., whether the new command is defined or the existing one is retained (therefore, **\providecommand{}{}** will not be discussed any more in this book).

[1] A new short command can defined to represent conveniently a long command or a combination of commands or even a piece of texts, if it is to be used repeatedly in a document.

D. Datta, *LaTeX in 24 Hours: A Practical Guide for Scientific Writing*,
DOI 10.1007/978-3-319-47831-9_13

13.1.1 New Commands Without Argument

It is discussed in §1.5.1 on page 5 as well as seen in previous Hours that some commands work on their own, while others ask a user to provide some inputs. Table 13.1

Table 13.1 Defining new commands without argument

Definition of new command (in the preamble)	Meaning
`\newcommand{\bs}{$\backslash$}`	'`\bs`' to print '\'
`\newcommand{\xv}{\mbox{\boldmath$x$}}`	'`\xv`' to print '$\boldsymbol{x}$'
`\newcommand{\veps}{\ensuremath{\varepsilon}}`	'`\veps`' to print 'ε'
`\newcommand{\cg}{\it Center of Gravity\/}`	'`\cg`' to print '*Center of Gravity*'

shows how a new command without any argument (i.e., without any input from a user) can be defined in the preamble and its meaning when used in the body of a document. The `\bs` command is defined to represent the math-mode `\backslash` command. However, it is applicable in text-mode only. If used in math-mode, the first `$` sign in its definition will quit the math-mode. To alleviate such errors, the inline math-mode may be generated through `\mbox{}`, which works in both text-mode and math-mode. Another option is to use `\ensuremath{}`, which always processes its argument in math-mode regardless the mode in which it is used. The applications of `\mbox{}` and `\ensuremath{}` are shown in Table 13.1 in the definitions of the `\xv` and `\veps` commands, which also show the representation of a combination of some existing commands by a single new command. On the other hand, the `\cg` command in Table 13.1 shows that, not only existing commands but a piece of normal texts can also be represented by a new command.

Note that, if ended by an alphabet, a user-defined new command (like `\bs`, `\xv`, `\veps` and `\cg` in Table 13.1) is also to be terminated by `\␣` in order to protect the trailing blank space. Without trailing `\␣`, the texts next to a command would be printed in continuation of the command, e.g., '`\xv␣is a vector`' will print '$\boldsymbol{x}$is a vector' (without any gap between '$\boldsymbol{x}$' and 'is'), while '`\xv\␣is a vector`' will print '$\boldsymbol{x}$ is a vector'.

13.1.2 New Commands with Mandatory Arguments

In the case of presenting a particular scenario repeatedly, a general command would be a more preferred one, which can be applied to different cases by changing the requirements of a user. For example, the `\xv` command of Table 13.1 is fixed to print `x` as a vector. If `y` is also to be presented as a vector in some cases, it will require to define another similar command, say `\yv`. This will not only increase the size of the preamble, but it would be difficult also to remember the commands defined for different cases. Therefore, instead of case-based fixed commands like `\xv` and `\yv`, it would be convenient to define a single command with some arguments to take the requirements of a user. Such a new command is defined through `\newcommand{}[]{}` as `\newcommand{newc}[n]{..{#1}..{#2}..{#n}..}`, where `newc` is the new command to be defined, `n` is the number of arguments (a command can have a maximum of 9

arguments), and #1, #2, …, #n are the serial numbers of the arguments. Although the arguments need not appear in order in the definition of a new command, they must be supplied in order during application, each argument in separate {}.

A number of examples of defining new mathematical commands having arguments are shown in Table 13.2. The definitions of the new commands may be

Table 13.2 Definition and application of new commands with mandatory arguments

Definition of new command (in the preamble)	Meaning
\newcommand{\vctr}[1]{\mbox{\boldmath{$#1$}}}	#1 as a vector.
\newcommand{\pde}[2]{\ensuremath{% \frac{\partial #2}{\partial #1}}}	Partial derivative of #2 w.r.t. #1.
\newcommand{\ode}[2]{\ensuremath{\frac{d#2}{d#1}}}	Ordinary derivative of #2 w.r.t. #1.
\newcommand{\oded}[2]{\ensuremath{% \frac{d^2#2}{d#1^2}}}	Second order OD of #2 w.r.t. #1.
\newcommand{\odp}[2]{\ensuremath{\frac{d}{d#1}(#2)}}	OD of (#2) w.r.t. #1.
\newcommand{\intg}[2]{\ensuremath{\int(#2)\,d#1}}	Integration of (#2) w.r.t. #1.
\newcommand{\dint}[4]{\ensuremath{% \int_{#3}^{#4}(#2)\,d#1}}	Integration of (#2) w.r.t. #1 from #3 to #4.
\newcommand{\lmt}[4]{\ensuremath{% \lim_{#3\to #4}\frac{#1}{#2}}}	Limit of #1/#2 for #3→#4.

SN	LaTeX input	Output
1	\vctr{x}.\vctr{y} = z	$\boldsymbol{x}.\boldsymbol{y} = z$
2	\pde{y}{x}	$\frac{\partial x}{\partial y}$
3	\ode{y}{x}	$\frac{dx}{dy}$
4	\oded{y}{x}	$\frac{d^2x}{dy^2}$
5	\odp{y}{x^2+3xy-5}	$\frac{d}{dy}(x^2+3xy-5)$
6	\intg{x}{x^5+4x^2-10}	$\int(x^5+4x^2-10)\,dx$
7	\dint{p}{p^3q+5pq-q}{0}{3}	$\int_0^3(p^3q+5pq-q)\,dp$
8	\dint{p}{p^3q+5pq-q}{v}{}	$\int_v(p^3q+5pq-q)\,dp$
9	\lmt{x^2+3x-10}{x-2}{x}{2}	$\lim_{x\to 2}\frac{x^2+3x-10}{x-2}$

difficult to understand, which is cleared through their applications. The \, command is used between (#2) and d#1 in the definitions of \intg and \dint for maintaining a small gap before a differential, which may be viewed before dx or dp in applications 6–8. It is also to be noted that if there is nothing to provide against a mandatory argument of a command, it can be left just by empty {}, which is demonstrated in application 8 by keeping the upper limit of \dint empty. In this way, \dint can be used for indefinite integral also, by leaving its last two arguments empty.

Tables 13.1 and 13.2 show the process of defining a new command to represent an existing command or a combination of existing commands, or even a piece of normal texts. The \newcommand{}{} can also be used to reduce the number of arguments of an existing command. For example, if texts in red color are to be produced repeatedly, as done in this book (for online version), a shorter command, say \tred, may be defined to replace the repeated use of the long \textcolor{red}. In that case, instead of \textcolor{red}{atext}, just \tred{atext} can be used for printing atext in red color.

13.1.3 New Commands with Optional Arguments

The arguments of all the new commands of Table 13.2 in §13.1.2 are mandatory. An optional argument can also be assigned to a user-defined new command[2]. Consider the case of \xv defined in Table 13.1, which prints x as a vector. The same \xv command with an optional argument can be used to print another letter also as a vector, instead of using \vctr{} shown in Table 13.2. A new command with an optional argument is defined in a similar way as shown in Table 13.2, but with an additional optional argument to \newcommand{}[]{}, i.e., through \newcommand{}[][]{} as \newcommand{newc}[n][farg]{..{#1}..{#2}..{#n}..}, where farg is the default first argument (which is optional) of the new command newc. Table 13.3 shows the

Table 13.3 Definition and application of new commands with optional arguments

Definition of new command (in the preamble)	Meaning
\newcommand{\xv}[1][x]{\mbox{\boldmath{$#1$}}}	Optional **#1** or default x as a vector.
\newcommand{\drv}[2][y]{\ensuremath{\frac{d}{d#1}(#2)}}	Ordinary derivative of **(#2)** w.r.t optional **#1** or default y.

SN	LaTeX input	Output
1	\xv␣is a vector.	$\boldsymbol{x}$ is a vector.
2	\xv[y] is also a vector.	$\boldsymbol{y}$ is also a vector.
3	\drv{x}	$\frac{d}{dy}(x)$
4	\drv[x]{\sin x}	$\frac{d}{dx}(\sin x)$

definitions and applications of two new commands, each with an optional argument. The \xv command has only one argument, which is optional. By default \xv prints $\boldsymbol{x}$, otherwise its optional argument if provided, say \xv[P] will print $\boldsymbol{P}$ (an optional argument is provided in []). On the other hand, the \drv command has two arguments – the first one is optional and the second one is mandatory. In the absence of the optional argument, \drv differentiates its mandatory argument with respect to default y, otherwise with respect to its optional argument if provided.

13.2 Redefining Existing Commands*

In some applications, the default style of an existing command may need to be altered, which is done by redefining the command[3]. An existing command is redefined in the preamble through the \renewcommand{rcom}[n]{astyle} command, where rcom is the existing command to be redefined, and astyle is its opted new style in terms of n number of arguments. For example, \renewcommand{\labelitemi}{{\small$\vartriangleright$}} replaces the bullet marking of

[2] Only one optional argument is permitted to a user-defined new command.

[3] An existing command can be redefined to alter its default style.

the items with '▷' under the first level of the itemize environment. Some heading commands generate their label-words, like \chapter{} generates 'Chapter' before its serial number. Such default label-words are listed in Table 13.4, which can be altered

Table 13.4 Heading commands and their default label-words

Heading Command	Default label-word	Heading Command	Default label-word
\abstractname	Abstract	\indexname	Index
\appendixname	Appendix	\listfigurename	List of Figures
\bibname	Bibliography	\listtablename	List of Tables
\chaptername	Chapter	\partname	Part
\contentsname	Contents		

through \renewcommand{}[]{}, e.g., \renewcommand{\chaptername}{Unit} for replacing 'Chapter' by 'Unit', or \renewcommand{\abstractname}{Summary} for replacing 'Abstract' by 'Summary'. Many more examples of redefining existing commands have already been provided in previous Hours, specially in Hours 4–6, 8, and 9.

Note that the first argument, if any, of a redefined command also can be made optional in a similar way as that of a new command stated in §13.1.3, i.e., through \renewcommand{rcom}[n][farg]{astyle} with farg as the default first argument of rcom.

Also note that the command to be redefined cannot be repeated in the second argument of \renewcommand{}{}, e.g., \renewcommand{\alpha}{Symbol-\alpha} is not permitted. In such a requirement, a command may be redefined in one of the following two processes:

1. First, save the existing command by another name, and then redefine the former in terms of the latter. As an example, first saving \textcolor as \oldtextcolor using '\let\oldtextcolor\textcolor', \textcolor then can be redefined as \renewcommand{\textcolor}[2][red]{\oldtextcolor{#1}{#2}} to print its argument in default red color or in the supplied optional color, i.e., \textcolor{atext} to print atext in red color and \textcolor[acol]{atext} to print atext in acol color.
2. First, obtain the LaTeX's internal coding of the existing command, and then redefine the command in terms of that coding. For example, \sigma can be redefined as \renewcommand{\sigma}{\mbox{\boldmath{$\mathchar"11B$}}} in terms of its internal coding '\mathchar"11B' so as to print σ by \sigma. LaTeX's internal coding of a command can be obtained by using \show, e.g., the use of \show\sigma somewhere in the LaTeX input file will pause its compilation displaying '> \sigma=\mathchar"11B', which means that the internal coding of \sigma is \mathchar"11B.

13.3 Defining New Environments

Like new commands discussed in §13.1 on page 125, user-specific new environments can also be defined. A new environment is usually defined in the preamble in terms of an existing environment for obtaining its content in a slightly modified pattern. Moreover, a completely new pattern can also be obtained as per requirement.

13.3.1 New Environments Without Argument

A new environment can be defined as `\newenvironment{nenv}{cstart}{cend}`, where `nenv` is the new environment, and `cstart` and `cend` are, respectively, the commands for starting and ending the new environment. Table 13.5 shows the

Table 13.5 Definition and application of new environments without argument

LaTeX input	Output

`\documentclass{article}`
`\newenvironment{itememm}%`
`{\begin{itemize}\em}{\end{itemize}}`
`%`
`\newenvironment{boxednote}{\begin{center}\em%`
`\begin{tabular}{|p{0.8\textwidth}|}\hline}%`
`{\\ \hline\end{tabular}\end{center}}`
`%`
`\begin{document}`
`\begin{itememm}`
`\item Emphasized items.`
`\item Modified itemize environment.`
`\end{itememm}`
`%`
`\begin{boxednote}`
`This is a new environment for ...`
`\end{boxednote}`
`\end{document}`
	• *Emphasized items.*
	• *Modified itemize environment.*
	This is a new environment for producing important notes and observations in emphasized fonts inside a box.

definitions and applications of two new environments. The first new environment is `itememm`, which is defined by modifying the existing **itemize** environment for producing its contents in emphasized fonts. It is done through **\em** inserted in the second argument of **\newenvironment{}{}{}**. Not only **\em**, other commands including a new one or a redefined one can also be added for obtaining a desired pattern. For example, the bullet marks (•) in Table 13.5 can be replaced by check-mark (✓) by defining the environment as '**\newenvironment{itememm}{\begin{itemize}\renewcommand{\labelitemi}{\checkmark}}{\end{itemize}}**', where **\labelitemi** is redefined as **\checkmark** (refer §6.1.2 on page 53 for detail).

The second new environment in Table 13.5 is `boxednote`, defined by combining two existing environments for producing emphasized notes in a box. The new environment employs the **tabular** environment for producing a single-column table as

a box, which is horizontally center aligned through the **center** environment. Finally, the **\em** command is used for emphasizing the contents of the new environment.

13.3.2 New Environments with Arguments

Similar to new commands discussed in §13.1.2 on page 126, a new environment can also be provided with some mandatory arguments, in which case it is to be defined through **\newenvironment{**nenv**}[**n**]{**cstart**}{**cend**}**, where n is the number of arguments to be provided by a user (a maximum of nine arguments can be provided). For example, in order to put a boldface title to the contents of the boxednote environment, the first line of its definition in Table 13.5 can be replaced with '**\newenvironment{**boxednote**}[**1**]{\begin{center}{\bf #**1**} \em**', and then the environment can be started in the **document** environment as **\begin{**boxednote**}{**atitle**\\}**, where atitle is the title of the contents to be printed in boldface fonts in a new line (it will be printed in a new line because of the line break command ****).

Further, an optional argument is also permitted to a user-defined environment. As in the case of new commands stated in §13.1.3 on page 128, a new environment with an optional argument is defined with an additional optional argument to **\newenvironment{}[]{}{}**, i.e., through **\newenvironment{}[][]{}{}** as **\newenvironment{**nenv**}[**n**][**farg**]{**cstart**}{**cend**}**, where farg is the default first argument (which is optional) of the new environment nenv. As an example, consider again the above case of the boxednote environment. In order to put an optional title to the contents of boxednote, the first line of its definition in Table 13.5 can be replaced with '**\newenvironment{**boxednote**}[**1**][]{\begin{center}{\bf #**1**}\em**'. If the environment is now started simply as **\begin{**boxednote**}**, its contents will be printed without any title because of the empty second optional argument in its definition. However, if it is started as **\begin{**boxednote**}[**atitle**\\]**, the optional argument atitle will be printed in boldface fonts in a new line as the title of the contents.

13.3.3 Theorem-Like Environments

Many mathematical documents include theorems and other theorem-like structures, such as definitions, lemmas, and propositions. A non-mathematical document may also contain similar structures, like principles, laws, assumptions, etc. For defining any such environment, LaTeX provides the **\newtheorem{**akey**}{**nenv**}[**aunit**]** command, where nenv is the name of the new environment and akey is its keyword, while optional aunit is the name of the sectional unit (like **chapter** or **section**) based on which the environment is to be numbered. The definitions of some of such environments are shown in Table 13.6 on the next page with or without an optional argument. Each theorem-like environment is numbered, preceded by the name of the environment. Numbering is done according to the optional argument provided in the

Table 13.6 Defining theorem-like environments

Definition	Meaning
`\newtheorem{thm}{Theorem}[chapter]`	'thm' for writing chapter-wise 'Theorem'
`\newtheorem{dfn}{Definition}[chapter]`	'dfn' for writing chapter-wise 'Definition'
`\newtheorem{cor}{Corollary}[section]`	'cor' for writing section-wise 'Corollary'
`\newtheorem{lem}{Lemma}[section]`	'lem' for writing section-wise 'Lemma'
`\newtheorem{prop}{Proposition}`	'prop' for writing 'Proposition'
`\newtheorem{prf}{Proof}`	'prf' for writing 'Proof'
`\newtheorem{princ}{Principle}`	'princ' for writing 'Principle'
`\newtheorem{rul}{Rule}`	'rul' for writing 'Rule'

Table 13.7 Application of the user-defined '`Definition`' environment

LaTeX input	Output

`\begin{dfn}[\bf Center of Mass]\label{dfn-cm}`
`This is the point at which the entire mass of a body of uniform density can be assumed to be concentrated.`
`\end{dfn}`
`%`
`\begin{dfn}{\bf Center of Gravity:}`
`This is the point though which the resultant of the gravitational forces of all elemental weights of a body acts.\label{dfn-cg}`
`\end{dfn}`
`%`
`Definition~\ref{dfn-cm} defines center of mass, while Definition~\ref{dfn-cg} ...`
	Definition 13.1 (Center of Mass) *This is the point at which the entire mass of a body of uniform density can be assumed to be concentrated.*
	
	Definition 13.2 Center of Gravity: *This is the point though which the resultant of the gravitational forces of all elemental weights of a body acts.*
	
	Definition 13.1 defines center of mass, while Definition 13.2 defines center of gravity.

definition of the environment, otherwise globally if no option is provided. Table 13.7 shows an example of the `Definition` environment, which is defined in Table 13.6 with the keyword `dfn` and optional argument **chapter**.

Like any other numbered item, a theorem-like environment can also be labeled and referred through a keyword. An environment can be labeled anywhere within its body. This is shown in Table 13.7, where the first `dfn` environment is labeled at the very beginning of its body and the second one is labeled at the bottom of its body.

On the other hand, a theorem-like environment can be provided a title also. It is to be provided after **\begin{}** as an optional argument in **[]** or explicitly in **{}**. Both the styles are shown in the two `dfn` environments in Table 13.7. Notice that, in the case of the optional argument, the title is printed in a pair of parentheses.

Further, it may be noticed in Table 13.7 that the contents of a theorem-like environment is printed in a new paragraph. Before defining such an environment through **\newtheorem{}{}[]** (refer Table 13.6), its style can be controlled using the **\theoremstyle{}** command defined in the **amsthm** package, e.g., **\theoremstyle{break}** for starting in a new line, or **\theoremstyle{plain}** for continuing in the same line. The **amsthm** package also defines the starred form of **\newtheorem{}{}**, i.e., **\newtheorem*{}{}** (without any optional argument), for producing unnumbered theorem-like environments. The **\newtheorem*{}{}** command may be useful for defining an environment which will be created one time only. It may also be useful if an environment is to be identified by a particular name rather than by a serial number, e.g., the definition of '`Center of Gravity`' as shown in Table 13.7.

13.3.4 *Floating Environments for Textual Materials**

Materials of some items, like a table or figure, are not desired to be split over pages. If the remaining blank space on the current page is not sufficient to fit such an item, LaTeX floats (shifts) it to a convenient place, such as the top or bottom of the current page or even on the next page. However, textual materials, like algorithms and computer programs (codings), are split over pages to avoid partially filled pages. Such materials can also be forced through a floating environment to appear together on a single page. For tables and figures, LaTeX provides two standard floating environments, namely **table** and **figure** respectively, which are discussed in Hours 7–10. Similar floating environments may be defined for textual materials also.

The **float** package provides the **\newfloat{}{}{}[]** command for defining a new floating environment in the preamble as **\newfloat{**`afloat`**}{**`vpos`**}{**`extn`**}[**`unit`**]**, where mandatory `afloat` is the name of the floating environment to be defined (e.g., `algorithm` or `program`), `vpos` is its vertical positioning (**h**, **b** or **t**, or a combination of them), `extn` is the extension of the auxiliary file for storing the captions of the floats (say, `flt`), and optional `unit` is to specify how the float will be numbered (e.g., **section** for section-wise numbering, or **chapter** for chapter-wise numbering). Some provisions are also available with floating environments, which are the following:

1. By default the first argument of **\newfloat{}{}{}[]** becomes the label-word for the caption of the defined float. It can be altered using **\floatname{**`afloat`**}{**`clabel`**}** (after **\newfloat{}{}{}[]**) for labeling the caption of the `afloat` environment by `clabel`.
2. The **\floatstyle{}** command can be used before **\newfloat{}{}{}[]** for defining the style of a float. The permissible styles (i.e., the argument of **\floatstyle{}**) are **plain**, **boxed** and **ruled**. The **plain** style prints the caption below the float regardless the position of the **\caption{}** command in the floating environment. The **boxed**-style produces the float in a box and prints the caption below the box. On the other hand, the **ruled**-style prints the caption above the float and also puts it between two horizontal lines. Moreover, another horizontal line is produced below the float.
3. By default a maximum of three floats are permitted on a single page. The number can be changed by redefining the counter **totalnumber** through the **\setcounter{}{}** command, e.g., **\setcounter{totalnumber}{**`10`**}** for allowing a maximum of 10 floats on a single page, if the available space of a page is sufficient for that. This counter is equally applicable to the standard **table** and **figure** floating environments.
4. Analogous to the **\listoftables** and **\listoffigures** commands used for producing lists of tables and figures respectively (§16.1 on page 153 discusses in detail), the **\listof{}{}** command can be used for producing the list of floats of a given type, e.g., **\listof{**`program`**}{**`List of Computer Programs`**}** will produce a list of `program`-type floats under the heading '`List of Computer Programs`'.

Table 13.8 on the next page shows the process for defining and applying two new user-defined floating environments, namely `algorithm` and `program`, for presenting algorithms and computer programs, respectively. Through the **\floatstyle{}**, **\newfloat{}{}{}[]**, and **\floatname{}{}** commands (in order), the **ruled**-style `algorithm`

Table 13.8 Floats for presenting algorithms and computer programs

LaTeX input

```
\documentclass[11pt,a4paper]{article}
\usepackage{float}
\floatstyle{ruled}
\newfloat{algorithm}{hbt}{alg}[section]
\floatname{algorithm}{Algorithm}
\floatstyle{boxed}
\newfloat{program}{hbt}{prg}[section]
\floatname{program}{Program}
%
\begin{document}
...
\section{Finding the maximum}
The main steps ... Algorithm~\ref{algo:max}.
\begin{algorithm}
\caption{Maximum of $n$ data points.}
\label{algo:max}
\begin{enumerate}
\item Read the number of data points $n$.
\item Read the data point $a_i$;
      $i=1$ to $n$.
\item Set {\it max} $=a_1$.
.
.
.
\end{enumerate}
\end{algorithm}
%
Algorithm~\ref{algo:max} is coded in the
{\tt C} computer ... Program~\ref{prog:max}.
\begin{program}
\caption{Maximum of $n$ data points.}
\label{prog:max}
\begin{verbatim}
#include<stdio.h>
#include<math.h>
int main()
{
    int n, a[101];
    int i, max;

    printf("Number of points = ");
    .
    .
    .
    return(0);
}
\end{verbatim}
\end{program}
...
\end{document}
```

Output

5 Finding the maximum

The main steps for finding and printing the maximum from a given data set of n points are shown in Algorithm 5.1.

Algorithm 5.1 Maximum of n data points.

1. Read the number of data points n.
2. Read the data point a_i; $i = 1$ to n.
3. Set $max = a_1$.
4. If $max < a_i$, set $max = a_i$; $i = 2$ to n.
5. Print max as the maximum of given n number of data points.

Algorithm 5.1 is coded in the C computer programming language, which is shown here in Program 5.1.

```
#include<stdio.h>
#include<math.h>
int main()
{
    int n, a[101];
    int i, max;

    printf("Number of points = ");
    scanf("%d", &n);
    for(i = 1; i <= n; i++)
    {
        printf("a[%d] = ", i);
        scanf("%d", &a[i]);
    }

    max = a[1];
    for(i = 2; i <= n; i++)
        if(max < a[i]) max = a[i];

    printf("\nLargest value = ");
    printf("%d\n", max);

    return(0);
}
```

Program 5.1: Maximum of n data points.

environment is defined with labeling its captions by `Algorithm` and storing them in an auxiliary file of `alg` extension. Similarly, the **boxed**-style `program` environment is defined with labeling its captions by `Program` and storing them in an auxiliary file of `prg` extension. Both the environments are opted to be numbered section-wise, and positioned here (**h**), bottom of the page (**b**) or top of the next page (**t**) as per the situation at hand. Then the applications of the environments are shown in the body of the LaTeX input file. The computer coding in the `program` environment is inserted

through the **verbatim** environment for printing the contents, as inserted, ignoring the LaTeX mode (§18.5 on page 173 discusses verbatim texts in detail). Note that, as shown in Table 13.8, a float environment can also be labeled and referred like any other environment.

13.4 Redefining Existing Environments*

It is stated in §13.3 on page 130 that a new environment can be defined in terms of an existing environment for obtaining its contents in a slightly modified pattern. Instead of creating a new environment, an existing environment can also be redefined for modifying its effect. However, redefinition of an existing environment will change its effect in the entire document. Hence, instead of redefining an existing environment, a new one should be defined, if changes are required only in certain portions of a document.

Similar to defining a new environment, an existing environment is redefined through the **\renewenvironment{**`nenv`**}{**`cstart`**}{**`cend`**}** command, where `nenv` is the name of the existing environment which is to be redefined, and `cstart` and `cend` are, respectively, the new commands for starting and ending the environment. Consider an example that a nested list of unnumbered items is to be prepared by eliminating inter-item spacing as well as emphasizing the items of every alternate loop. This can be done by redefining the **itemize** environment as shown in Table 13.9 by reproducing

Table 13.9 Redefining the unnumbered listing environment **itemize**

LaTeX input	Output
\documentclass[11pt, a4paper]{article} **\usepackage{paralist}** **\renewenvironment{itemize}%** **{\em\begin{compactitem}}{\end{compactitem}}** **%** **\begin{document}** **\begin{itemize}** **\item** India **\begin{itemize}** **\item** Assam **\begin{itemize}** **\item** Sonitpur **\begin{itemize}** **\item** Tezpur **\item** Dhekiajuli **\item** Balipara **\end{itemize}** **\item** Kamrup **\item** Cachar **\end{itemize}** **\item** Bihar **\item** Punjab **\end{itemize}** **\item** Pakistan **\item** Sri Lanka **\end{itemize}** **\end{document}**	• *India* – Assam ∗ *Sonitpur* · Tezpur · Dhekiajuli · Balipara ∗ *Kamrup* ∗ *Cachar* – Bihar – Punjab • *Pakistan* • *Sri Lanka*

the list given in Table 6.7 on page 54. In this example, the **itemize** environment is redefined as the **compactitem** environment, which is defined in the **paralist** package. Further, the **\em** command is used in the redefinition of **itemize** for obtaining emphasized texts.