CHAPTER 15: CUSTOMIZING LaTeX

NOAH C. RUDERMAN

Abstract. We will review
- User-defined commands
- User-defined environments
- A custom command file
- Numbering and measuring
- Custom lists

User-defined commands, also called macros, can make inserting text or equations you use often easier. For example, if you use the equation

$$\int_0^{\infty} P(X = x)dx \tag{1}$$

often, you may redefine it as a command. To create a user defined command, we use the command `\newcommand`{*command 1*}{*command 2*} in the header. *command 2* is the command we wish to make the equivalent of typing *command 1*. So if we wanted to typeset equation (1) with the shorter command `\Px`, we would insert the following into the header: `\newcommand{\Px}{\int_0^{\infty} P(X = x) dx}` As proof we use the user-defined command below

$$\int_0^{\infty} P(X = x)dx \tag{2}$$

There are a few things to keep in mind:

(1) The first argument must not be a command that is already in use
(2) The second argument must be spelled correctly, but LaTeX will only give you an error when you try to use it if this is the case.
(3) Macros in LaTeX work the same way as in C, so command declarations will need an extra pair of braces.

In the above example when we defined the command `\Px`, we see that it is a math formula so we can only use it in a math environment. Alternatively, we could have defined `\Px` to be `$\int_0^{\infty} P(X = x) dx$` so that it could be used in a text environment, but then this would prohibit use in a math environment without `$` delimiters. There is a solution around this. We can use the `\ensuremath{...}` command so that the `\Px` command works in both environments. We would change the original user defined command to the following: `\newcommand{\Px}{\ensuremath{\int_0^{\infty} P(X = x) dx}}`. Now `$\Px$` and `\Px` typeset as $\int_0^{\infty} P(X = x)dx$.

We can also make user-defined commands that take arguments. For example, if we wanted to define a `\pwr`{*base*}{*exponent*} command that prints $\text{base}^{\text{exponent}}$, we would put the following in the header:
`\newcommand{\pwr}[2]{\ensuremath{{\text{#1}}^{\text{#2}}}}`. We use it in the following: $(\text{argument 1})^{(\text{argument 2})}$. To define a command with arguments,

we insert a number enclosed in brackets after the first set of braces. This number may take any value from 1-9 and represents the number of arguments that the command takes. In the second pair of braces, we type 1, 2, ... n to insert the value of the 1$^{\text{st}}$, 2$^{\text{nd}}$, or n$^{\text{th}}$ argument.

We can make the first command optional and provide a default value for the optional argument. Suppose we would like a command that accepts two arguments (let's assume they're a and b) and typesets $a_1 + a_2 + \cdots + a_b$. We could put the following in the header:
`\newcommand{\seq}[2][n]{\ensuremath{#2_1 + #2_2 + \cdots + #2_{#1}}}`
This says that the `\seq` command takes two arguments, that the first argument is optional, and that the default value for the first arguement is n. Now that the first argument is optional, we enclose it in brackets `[ ]` rather than braces { }. We omit the brackets, rather than providing them with no value to tell LaTeX that we would like to use the default value.

$$\begin{aligned} \texttt{\textbackslash seq\{x\}} &\longrightarrow x_1 + x_2 + \cdots + x_n \\ \texttt{\textbackslash seq[]\{x\}} &\longrightarrow x_1 + x_2 + \cdots + x \\ \texttt{\textbackslash seq[z]\{x\}} &\longrightarrow x_1 + x_2 + \cdots + x_z \end{aligned}$$

We may also redefine commands, although it would be used sparingly. For example, the QED symbol is typeset as `\qedsymbol` ■. If we would like a solid black square ■ rather than the hollow square, we may add the following to the header: `\renewcommand{\qedsymbol}{\ensuremath{\blacksquare}}`. Now `\qedsymbol` appears as ■.

This is somewhat unrelated, but the command `\linewidth` is the horizontal spacing of a given line in the current format.

We can also create user defined environments. Here is a user-defined pseudo environment you've been using:

```
\phantom{\qquad}\textbf{a.}
\begin{minipage}[t]{11 cm}
   Your text here.
\end{minipage}
\\[1ex]
```

This pseudo-environment is the following:

a. Your text here.

Here's how you would redefine the environment to do such a thing:

```
\newlength{\cgap}
\settowidth{\cgap}{\qquad \textbf{x. }}

\newlength{\cwidth}
\setlength{\cwidth}{\textwidth}
\addtolength{\cwidth}{-\cgap}

\newenvironment{cpart}[2][\cwidth]
   {\\ \phantom{\qquad}\textbf{#2. }\begin{minipage}[t]{#1}}
   {\end{minipage} \\}
```

First we define a custom width called `\cwidth` for formatting purposes. Then we use the `\newenvironment` command. There are three mandatory arguments and two optional arguments. The first mandatory argument is the name of the environment, so that we may invoke with with `\begin`{*environment name*}. Then there are two optional arguments; The first optional argument specifies the number of arguments that the environment takes, so we may now invoke it with `\begin`{*environment name*}{*argument 1*}{*argument 2*}... The second optional argument implies that the first optional argument is used and makes *argument 1* optional. The value of the second optional argument is the default value. As always, we place this all in the header. Now our environment invocation takes the form:
`\begin`{*environment name*}[*argument 1*]{*argument 2*}...
or `\begin`{*environment name*}{*argument 2*}... (*argument 1* takes default value)
The second mandatory argument is what is inserted when we invoke our new environment with the `\begin`{*environment name*}... command and the third mandatory argument is what gets inserted when we end our environment with the
`\end`{*environment name*} command.

This is what the environment looks like:

a. This is part (a), and as I type we will see if the formatting is correct by comparing where the line breaks off in this environment to outside of the environment.

b. This is part (b), and we can see the implicit spacing between the two environments. Here is an equation that is part of the environment:

$$\int_0^\infty P(X = x)dx$$

Instead of putting all of the user-defined commands in the header, you may choose to put them all in a custom command file. The program extension in the book for the command file is `.sty`, and I am unsure exactly why that is so. You place everything in the command file as normal, but you must place the `\endinput` command at the end of the file. To include the file in the document, you use the `\usepackage`{...} command. All of the commands used in this file, for example, were moved into a file named `ch15CommandFile.sty` and imported with the `\usepackage{ch15CommandFile}` command in the header.

Counters are automatically handled by LaTeX but sometimes custom formatting by the user is necessary. Examples of counters are the numbering of a theorem, section, equation, and page numbers. We can set extant counters with the `\setcounter`{*counter*}{*value*} command. We can also get the value of a counter by the `\the`*`counter`* command. We see that the value of `\theequation` is 2. Now we type `\setcounter{equation}{999}` and the value of `\theequation` is 999. Now let's see what number we get when we type an equation:

$$\int_0^\infty P(X = x)dx \tag{1000}$$

And now the value of `\theequation` is 1000.

LaTeX has standard counters used frequently with the following names

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

We can choose to define our own counters with the `\newcounter`{*mycounter*} command, to be placed in the header file.

We can change the counter's appearance if we wish with the `\renewcommand`{`\the`*`counter`*}{*new_style*} command. The following styles are available:

Style	Command	Sample
Arabic	`\arabic`{*`counter`*}	1, 2, . . .
Lowercase Roman	`\roman`{*`counter`*}	i, ii, . . .
Uppercase Roman	`\Roman`{*`counter`*}	I, II, . . .
Lowercase Letters	`\alph`{*`counter`*}	a, b, . . . , z
Uppercase Letters	`\Alph`{*`counter`*}	A, B, . . . , Z

We can get creative in how we redefine counters. For example, the page numbers were redefined with the command `\renewcommand{\thepage}{(\arabic{page} or \Roman{page})}` The following commands might be of use: `\stepcounter`{*`counter`*} increments *`counter`* and sets all the counters that were defined with the optional argument counter to 0. `\addtocounter`{*`counter`*}{*`n`*} adds the value *`n`* to *`counter`*.

LaTeX has five absolute length units:

- `cm` centimeter
- `in` inch
- `pc` pica (1 pc = 12 pt)
- `pt` point (1 in = 72.27 pt)
- `mm` millimeter

and two relative ones:

- `em`, approximately the width of the letter M in the current font.
- `ex`, approximately the width of the letter x in the current font.

See chapter 10 for common length commands.

We can define a new length command in the header like so: `\newlength{\mylength}`. Now `\mylength` is interpreted as a length by LaTeX.

To set the length of a length command, we type `\setlength{\mylength}`{*`length`*}. Some things to note:

1. The first argument must be a length command, which is to say that it must include the backslash \.
2. The second argument have units or be a length command.

Another way we can set the length of a command is to use the `\settolength`{*`length command`*}{*`argument`*}. The last argument can be text or a math formula which is to be measured by LaTeX whose length is assigned to the length command.

We can adjust the length of a length command with the `\addtolength`{*`length command`*}{*`length`*} command. One thing to note here is that the last argument can take on negative values as well as positive ones. Not including a sign in front of the last command means that LaTeX assumes you are implying the use of a positive value. Also related to this command are the commands `\settoheight` and `\settodepth` which take the same arguments. Note: "depth" is the length from the bottom of the line to the bottom of the box, which matters for letters like g and q.

Let's take a look at our previous use of length commands:

```
\newlength{\cgap}
\settowidth{\cgap}{\qquad \textbf{x. }}

\newlength{\cwidth}
\setlength{\cwidth}{\textwidth}
\addtolength{\cwidth}{-\cgap}

\newenvironment{cpart}[2][\cwidth]
   {\\ \phantom{\qquad}\textbf{#2. }\begin{minipage}[t]{#1}}
   {\end{minipage} \\}
```

We see that the judicious use of length commands can solve problems that I used to find baffling.

It is possible to make custom lists, which addresses yet another problem I was having not very long ago. First, we need to discuss the relevant length commands for a list environment.

`\topsep`: this is most of the vertical space between the first item and the preceding text and also most of the space between the last item and the following text. The space aforementioned is also augmented by `\parskip`, the extra vertical space inserted between paragraphs.

`\parsep`: this is the space between paragraphs of the same item.

`\itemsep`: this is the space between items. Like `\topsep`, the actual gap is the sum of `\itemsep` and `\parsep`.

`\leftmargin`: this is the distance between the edge of the item box and the left margin of the page. I believe the item box is the text that follows `\item`.

`\rightmargin`: analog of `\leftmargin` entry for right margin.

`\labelwidth`: the width of the box that the optional argument for the `\item` command is typeset in.

`\itemindent`: the amount of space that the box of length `\labelwidth` is indented from the left margin.

`\labelsep`: the space separation between the box that contains the label name and the box containing the text that follows.

`\listparindent`: the length of the indentation of the second and subsequent paragraphs of an item.

Custom lists are created using the `list` environment, which is invoked as follows:

```
\begin{list}{default_label}{declarations}
    \item item1
    \item item2
    ...
\end{list}
```

`default_label` is the default label used for any items that do not specify their own label. *`declarations`* is the vertical and horizontal length commands and any other required parameters for the list.

Here's an example of a customized list (with counters):

Rule 1: We may use counters in our lists and we must specify that the counter will be used in the *`default_label`* argument.

Rule 2: If we use a counter and wish for it to be incremented automatically, we use the `\usecounter{`*`counter`*`}` command.

Strangely, I am unable to get rid of the initial indentation of the first paragraph of an item. If we happen to use a custom `list` environment often, we can define it as a new environment. We can also apply labels to the items, and what will be recorded is the value of the counter for that item.