Ph3 LaTeX Week 6:
Lists and tables in paragraph mode

Eric D. Black

August 19, 2021

1 Lists

1.1 Making simple lists

Very often you will want to make lists of things. If you want the items in the list numbered, use the *enumerate* environment. If you just want unnumbered bullet points, use *itemize*. Each new item starts with the \item command, like so.

```
\begin{enumerate}
    \item First you do this.
    \item Then you do that.
    \item Profit!!!
\end{enumerate}
```

1. First you do this.
2. Then you do that.
3. Profit!!!

You can nest these environments, and the sublists have different styles of counters.

1. First you do this.
2. Then you do that.
    (a) But don't forget to do this,
    (b) this,
    (c) and this first.
3. Profit!!!

This works up to four levels deep. Generally, if you need to make nested lists more than four levels deep, you need to re-evaluate your writing style.

1.2 Counters

At this point most latex tutorials will teach you how to customize the style of your counters. I don't know why you would want to do this (I never have), but if you really feel the need you can see any of the usual guides [1].

What I have found useful is making a running list, where you transition in and out of the enumerate environment, interspersing blocks of explanatory text, and having each new list pick up its numbering where the last one left off. I will now demonstrate.

1. First you do this.
2. Then you do that.

Now comes some crucial information you need to include in your document that allows the reader to understand the next step. This text comes *after* the ending of the previous *enumerate* environment and *before* the beginning of the next *enumerate* environment. For proper flow of the text, however, you want the numbering to pick up where it left off in the previous list, like so.

3. Profit!!!

This is more than just a cute trick. It reveals something deeper about how latex works, and that makes it worth learning even if you never have to intersperse text and list items.

1.3 Accessing and manipulating counter variables

As we have seen, latex numbers many different objects, sections, subsections, pages, equations, tables, etc. The compiler keeps track of these through variables called *counters*. There is a counter for each class of object, and at the beginning of each document all the counters are initialized to zero. When the compiler encounters a command to define a new one of these objects, it iterates that counter. For enumerated lists the counter is called *enumi*, and it is reset to zero every time you begin a new enumerate environment. Sublists are counted with the variables *enumii, enumiii,* and *enumiv*, for the second, third, and fourth levels, respectively. Other names and commands are listed below.

1.3.1 Built-in counter names

1. List counters
   - enumi
   - enumii
   - enumiii
   - enumiv
2. Document-structure counters

- part
- chapter
- section
- subsection
- subsubsection
- paragraph
- subparagraph
- page

3. Floats
   - equation
   - figure
   - table
4. Footnotes
   - footnote
   - mpfootnote

1.3.2 Commands to access counter values

You can access the values of these counter variables two ways. To get a formattable string you can display in text, use the command `\thecounter`, where *counter* is the name of the variable you want to access. For example, to find out the current value of the *page* counter, use the command `\thepage`. To get a numerical value you can pass to a function, use the command `\value{counter}`, where the argument *counter* is the name of the counter you want to access. The command \value{} does not produce a formattable string and thus cannot be used to produce printable output.

- `\thecounter` returns a formattable string displaying the value of *counter*. This can be displayed in output.
- `\value{counter}` returns the numeric value of *counter* for passing to a non-output command.

1.3.3 Commands to change counter values

- `\stepcounter{counter}` increments the value of *counter* by one.
- `\refstepcounter{counter}` increments *counter* and updates its value in the referencing mechanism so `\ref{label}` will reflect the correct value.
- `\addtocounter{counter}{amount}` adds *amount* to the value of *counter*.
- `\setcounter{counter}{number}` set the value of *counter* to *number*.

1.3.4 Creating new counters

- `\newcounter{newcountername}` defines a new counter with the name *newcountername* and sets its initial value to zero; usually goes in the document preamble.

You can set your new counter to reset to zero every time another counter increments by adding the other counter's name as an option after the *newcountername* argument. For example, if you want to reset your custom counter back to zero every time you start a new section you would define it like so.

```
\newcounter{newcountername}[section]
```

3 Exercises

Exercise 1: What happened to Section 2? How would you make an entire section disappear like I just did?

Exercise 2: Format a "broken" list like I did in Subsection 1.2. Set things up so you can end an enumerate environment, write a paragraph of conventional text, then begin a new enumerate environment and have its counter pick up where you left off in the previous enumerate environment.

4 Tables

We've covered the *array* environment in math mode. There is a similar environment called *tabular* that works in either math or paragraph mode, and it can be used to produce nice-looking tables with text, numbers, or formulae in cells and borders around those cells. Commands for spacing between cells and starting new lines are the same as in the array environment (& and \\), and the options for centering, right justifying, or left justifying a column are also the same.

Borders are produced by a combination of horizontal and vertical lines. The vertical lines are specified with vertical bars (|) in the position field, and horizontal lines are produced with the commands \hline and \cline{i-j}, where \hline produces a line across the entire table, and \cline{i-j} makes a partial line that only covers columns i through j.

Lamport has the best example of this I've ever seen [3], and I will reproduce it here without apology.

```
\begin{tabular}{||l|lr||}       \hline
gnats     & gram    &\$13.65 \\ \cline{2-3}
          & each    &    .01 \\ \hline
gnu       & stuffed &  92.50 \\ \cline{1-1} \cline{3-3}
emu       &         &  33.33 \\ \hline
armadillo & frozen  &   8.99 \\ \hline
\end{tabular}
```

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

Table 1: A table formatted using the *tabular* environment.

4.1 Multicolumn entries

You can make a single entry span multiple columns with the command

```
\multicolumn{n}{pos}{item}
```

where n is the number of columns to span, *pos* is the positioning (left, right, or center), and *item* is the entry you want in that multicolumn cell. For example, if you were out of armadillo you might modify Lamport's table to reflect that shortage like this.

```
\begin{tabular}{||l|lr||}       \hline
gnats     & gram    &\$13.65 \\ \cline{2-3}
          & each    &    .01 \\ \hline
gnu       & stuffed &  92.50
                \\  \cline{1-1} \cline{3-3}
emu       &         &  33.33 \\ \hline
armadillo & \multicolumn{2}{c||}{\emph{not avail.}} \\ \hline
\end{tabular}
```

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	*not avail.*	

Table 2: A two-column entry example.

4.2 Tables as floats

The *tabular* environment produces tables that cannot be split across pages, so it is best used in *floats, i.e.* figures or tables. In Week 3 we saw how the *figure* environment works, and we saw how it allowed us to number and reference our figures, and how to add captions and influence the position of the figure on the page. The *table* environment works exactly the same way, even up to including the same optional positioning commands (`[h!]`, etc.). You make a floating table by nesting the *tabular* environment inside the *table* environment.

If you have a long table that needs to be broken up between pages, use either the *tabbing* environment or the *longtable* package and environment of the same name. I don't want to cover longtable beyond just letting you know it exists, but I will give a brief introduction to the tabbing environment in the context of listing computer code, where it is most useful.

5 Formatting computer code

There are two ways to format computer code. The one I use the most is the *verbatim* environment. Anything printed inside this environment gets reproduced in the output exactly as it is typed. No commands are interpreted, and all special symbols come out just as they appear in the source code. There is no need to specially format these symbols. In the verbatim environment, a backslash is just a backslash. Verbatim also uses a monospaced typewriter font, so you can adjust your indentations manually.

The verbatim environment produces standalone code, just as the equation environment produces a standalone equation. For an inline verbatim environment use \verb{}. This one is a little special in that the delimiters can be any special symbol not used inside the argument. For example, if I want to format something with pointy brackets ({}) I would have to use some delimiter other than pointy brackets. Exclamation points work, as long as I don't have any inside the argument. For example, I format `\value{counter}` like this `\verb!\value{counter}!`.

The other way to list computer code is using the *tabbing* environment. This allows you to indent your code and line things up the way computer-science textbooks teach you is good form. I have never used it in a publication, but I'll include Lamport's example here just for completeness.

```
\begin{tabbing}
If \= it's raining          \\
   \> then \= put on boots, \\
   \>      \> take hat;     \\
   \> else \> smile.        \\
Leave house.
\end{tabbing}
```

If it's raining
  then put on boots,
       take hat;
  else smile.
Leave house.

As in the tabular environment, \\ starts a new line, but instead of & to delineate cells you have the indentation commands \> which move you to the next *tab stop*. The \= command defines where those tab stops are.

6 More exercises

Exercise 3: Format the table you created in Taylor, Problem 6.6 in your hello world latex document. Make three columns, one for the number of samples N, one for the probability of finding a value outside $t_{\text{sus}}\sigma$, and a third for the actual value of t_{sus}. Make a horizontal line to separate the headers from the values and vertical lines to separate the columns from each other. You can make borders or not, depending on what you think looks better.

Hint: To format the subscript on the t value correctly, use

```
t_{\mbox{sus}}
```

The `\mbox{}` command reverts latex back into temporary text mode inside math mode, and the main thing I use it for is formatting subscripts just like this. While it does produce a text mode, it is *not* paragraph mode. Rather, it is the *left-to-right* or LR mode that we talked about last week. The principal difference between paragraph and LR mode is that LR mode does not produce line or paragraph breaks.

Exercise 4: Show the source code you used to format the table in the previous problem in your output document. You can use either the *verbatim* environment or the *tabbing* environment, whichever you prefer.

References

[1] `https://www.overleaf.com/learn/latex/lists`

[2] `https://www.overleaf.com/learn/latex/Counters`

[3] Lamport, Leslie *LaTeX: A Document Preparation System User's Guide and Reference Manual 2ed.*, Addison-Wesley Publishing Company 1994.