A non-expert looks at a small TEX macro

David Walden

Introduction

I use TEX a lot, but I seldom dig deeper into how TEX works than I must in order to address the immediate writing project I am working on. However, once I think I have figured out something new, I like to write it up to help me be sure I understand it. In this piece I describe a simple LATEX macro I wrote, how the macro evolved, and what I learned along the way. Perhaps other intermediate users who have a similar incremental approach to increasing their capabilities to use TEX will find reading my account a short cut to understanding of their own.

My problem

In some documents I write, I use an extra blank line and an extra large letter on the first character of the first word of a paragraph to indicate a thought break.

Here is an example.

A couple of years ago, I wrote a simple LATEX macro to accomplish this:

```
\newcommand{\newthoughtgroup}[1]{%
  \bigskip\noindent{\Large #1}}
```

It was called as follows:

```
\newthoughtgroup{H}ere is an example.
```

However, I didn't like having the first word of the paragraph in my LATEX file being split as in the above line. I wished the macro call could be

```
\newthoughtgroup{Here} is an example.
```

but still only make the first character of the first word larger.

Search and discovery

Therefore, I looked around for a way to have the whole first word be part of the macro argument — I had to look around since I didn't understand TEX macros well enough to be able to figure it out myself.

First approach. I discovered the following pair of macros on `comp.text.tex` (April 6, 1994) in a posting by Victor Eijkhout, who was answering a question about making the first letter of a word be upper case:[1]

```
\def\CapString#1{%
  \CapFirstLetter#1$} %assumes no $ in arg 1
\def\CapFirstLetter#1#2$%{
  \uppercase{#1}#2}
```

Without fully comprehending how Eijkhout's macros worked, I changed them as follows to accomplish my purpose:

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1$}
\def\BigFirstLetter#1#2$%{
  \bigskip\noindent{\Large #1}#2}
```

I suspect I am not alone among TEX user in blindly copying or converting something that already exists without much understanding of how it works.

Learning more. After using my version of Eijkhout's macros for a while, I decided to try to understand them in detail. So, I looked at chapter 20 of Knuth's *The TEXbook*;[2] in particular, I tried to understand from the first dangerous bend signs on page 203 to the first dangerous bend signs on page 204. The following is what I think I learned.[3]

First, I noted the difference between LATEX macro definitions and TEX macro definitions. My original LATEX macro listed above might be written as a TEX macro as follows:

[1] I've suddenly jumped to TEX style macro definitions instead of the LATEX form of macro definitions because that is what I found searching `comp.text.tex`, and for another reason that may become apparent.

[2] Addison Wesley, Reading, MA, 1986.

[3] I am not going to repeat the full explanation of a macro definition or how a macro finds its arguments when called; I'll just use what I learned to explain the macros I was working with.

```
\def\newthoughtgroup#1{%
  \bigskip\noindent{\Large #1}}
```

The TEX form of macro definition includes `\def`, followed by the new macro name (`\newthoughtgroup` in our case), followed by what Knuth calls the *parameter text* which in this case is `#1` indicating the macro has one *undelimited parameter*, and ending with the *replacement text* (`\bigskip\noindent{\Large #1}`). The call-time argument of an undelimited parameter is the first non-blank *token*,[4] or the tokens enclosed in matched braces, after the macro name.

This same format of TEX macro definition is used for the first macro below.

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1$}
\def\BigFirstLetter#1#2$%{
  \bigskip\noindent{\Large #1}#2}
```

The parameter text is `#1`, and the replacement text is `\BigFirstLetter#1$`. Thus, when the first macro is called with

```
\newthoughtgroup{Here}
```

the macro is *expanded* into its replacement text, which thus becomes `\BigFirstLetter Here$`.[5]

But the second macro's parameters specify a slightly different form of macro call. The first parameter (`#1`) is undelimited and, thus, the macro call's first argument is the first (non-blank) token or tokens enclosed in braces (as with the first macro). The second parameter, however, is *delimited* by the following `$` and, thus, the macro call's second argument is all the tokens from the end of the first argument to the `$`, i.e., to the delimiter.

Thus, when the first macro calls the second macro, that macro call (`\BigFirstLetter{Here$}`) finds its first argument to be `H` and its second argument to be `ere` with the `$` being discarded after matching. In turn, the call to `\BigFirstLetter` is replaced by

```
\bigskip\noindent{\Large H}ere}
```

producing the desired vertical space, no indentation, a big `H`, and normalsize `ere`.

Second approach. I happily used these macro definitions for a long time until I discussed them one day recently with Karl Berry. He pointed out that my version of Victor's formulation can be changed to remove that restriction on including `$` in the argument. He explained that the second argument's delimiter doesn't have to be a character; it can be an arbitrary control sequence (even an undefined control sequence), and he wrote down the following for me:[6]

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1\enddavesmacro}
\def\BigFirstLetter#1#2\enddavesmacro{%
  \bigskip\noindent{\Large #1}#2}
```

Third approach. That sounded like a good improvement, but then Karl said, "Personally, I would be inclined to a different approach, that has the benefit of being called without braces — which thus addresses your original reason for moving from a macro called with `\newthoughtgroup{H}ere`." He showed me the following definition for `\newthoughtgroup`:

```
\def\newthoughtgroup#1{%
  \bigskip\noindent {\Large #1}}
```

When called, for example, as

```
\newthoughtgroup Here is an example.
```

the argument that replaces the parameter (`#1`) is the `H`, i.e., the first non-blank token.[7]

Conclusion

As I started drafting this conclusion, it gradually dawned on me that the Third Approach TEX macro is the same as the TEX transliteration of my original LATEX macro ("Learning more" section), and perhaps my original LATEX macro ("My problem" section) also worked when called without braces:

```
\newthoughtgroup Here is an example.
```

[4] Tokens are described between exercises 7.2 and 7.3 on pages 38–39 of *The TEXbook*. As what the user typed is read into TEX, the letters, numbers, command names, etc., are stored as *tokens*. Tokens are internal representations of the characters in the input stream, with the notable exception that *control sequences* (e.g., `\bigskip`, `\def`, `\newthoughtgroup`) are each stored as single tokens. Macro definitions are stored as tokens, and macro calls are processed in terms of tokens.

[5] My macros are usually so simple that I can just think of the literal characters of the macro definition replacing the literal characters of the macro call in the sequence of characters that TEX reads, and so the definition `\newthoughtgroup` in this section originally looked funny to me. I wondered why the replacement text for `\newthoughtgroup{Here}` wasn't `\BigFirstLetterHere$` and then wondered why TEX didn't report that as an undefined control sequence. The answer, I believe, is that, as noted in footnote 4, TEX processes macros in terms of tokens, and the replacement text, `\BigFirstLetter#1$`, is manipulated as three distinct tokens: `\BigFirstLetter`, `#1`, and `$`.

[6] Victor also showed me a different formulation — one optimized for efficiency — that I will not try to explain in this note.

[7] Karl was not quite done yet. His final note was that if I was willing to stop trying to figure out macros like these, the "lettrine" package has support for many variations along the lines I desired. See `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=dropping` for mention of the package and `http://www.tex.ac.uk/tex-archive/macros/latex/contrib/lettrine/doc/demo.pdf` for a demonstration document.

It does — a bit of a startling conclusion for me.

There are two possible lessons here. Perhaps I originally should have posed my real problem to `comp.text.tex` rather than searching for "first letter of a string"; I might have been pointed in the right direction of understanding how TEX macro calls find their arguments. Or perhaps it paid to wander in some less-than-optimal directions; my journey of discovery was enlightening and relatively painless, and trying to explain it in writing definitely consolidated my knowledge — and I hope helped you.

Acknowledgements

I appreciate Victor Eijkhout's deep understanding of how the TEX program processes the TEX language (his book *TEX by Topic* has a comprehensive discussion of how TEX processes macros, `http://www.eijkhout.net/tbt/`) and also the deep understanding of Karl Berry and his suggestions as I prepared this paper.

Biographical note

David Walden is retired after a career as an engineer, engineering manager, and general manager involved with research and development of computer and other high tech systems. These days he does a lot of writing.

◇ David Walden
East Sandwich, MA
`www.walden-family.com/dave`