Glisterings: Longest string; Marching along; A blank argument; A centered table of contents

Peter Wilson

Plain as the glistering planets shine
When winds have cleaned the skies,
Her love appeared, appealed for mine,
And wantoned in her eyes.

Songs of Travel, Robert Louis Stevenson

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

The chief defect of Henry King
Was chewing little bits of string.

Cautionary Tales, Hillaire Belloc

1 Longest string

Romildo wrote to `comp.text.tex` saying that he tried to implement a macro for determining the longest string in a list but was having problems with the code [18]. Romildo's user view of the macro (`\Widest`) was like this:

```
\newdimen\mydimen
\def\Format#1{{\itshape\tiny #1}}
\Widest{\mydimen}{\Format}{Good,morning,world}
\the\mydimen
```

There were several responses, including ones from GL [8] and Heiko Oberdiek [16] who got into a bit of a discussion about their suggested solutions, partly because GL preferred the strings to look like multiple arguments (e.g., `{a}{bbb}{cc}`) and Heiko appeared to lean more towards a single argument with the strings being separated by commas (e.g., `(a,bbb,cc)`) .

GL suggested (I have used `\Widestg` for GL's macro and `\Widesth` for Heiko's to distinguish between them):

```
\makeatletter
\newskip\result
\def\Widestg#1#2#3\Widestg{% #1 = Format
  \setbox\z@\hbox{#1{#2}}%
  \ifdim\wd\z@>\result
    \result\wd\z@
    \edef\longest{#2}% % added by PW
    \def\flong{{#1{\longest}}}% % added by PW
  \fi
  \ifx\relax#2\else
    \Widestg{#1}#3\Widestg
  \fi}
\makeatother
...
\result=0pt
\Widest{\textbf}{one}{two}{three}\relax\Widest
\the\result \\
\longest\ \the\result\\  % added by PW
\flong\ \the\result      % added by PW
```

I added the code for `\longest` which contains the longest string and `\flong` to typeset it using the specified format. This code, applying the macro to the list `{one}{two}{three}`, results in:

26.13898pt
three 26.13898pt
three 26.13898pt

Heiko came up with a version that uses the kvsetkeys package [14] for parsing a comma-separated list where spaces at the beginning and end of an entry are ignored.

```
\usepackage{kvsetkeys}
\newcommand*{\Format}[1]{\textit{\tiny #1}}
\newlength\WidestResult
\makeatletter
\@ifdefinable{\Widesth}{%
  \def\Widesth#1#2(#3){%
    #1=\z@ % 0 pt
    \comma@parse{#3}{%
      \settowidth\dimen@{#2{\comma@entry}}%
      \ifdim#1<\dimen@
        #1=\dimen@
        \edef\longest{\comma@entry}% PW added
        \def\flong{#2{\longest}}%    PW added
      \fi
      \@gobble % ignore list entry argument
    }%
  }%
}
\makeatother
...
\Widesth{\WidestResult}{\Format}(Good,morning,
                                  world)
\the\WidestResult \\
\longest\ \the\WidestResult\\    % added by PW
\flong\ \the\WidestResult        % added by PW
```

Just as with `\Widestg` I added the `\longest` and `\flong` code. Note that the comma-separated list of strings is enclosed in parentheses and not braces. The result from Heiko's example is:

21.64417pt
morning 21.64417pt
morning 21.64417pt

Applying GL's macro to Romildo's example as:

```
\result=0pt
\Widestg{\tiny\textit}{Good}{morning}
                  {world}\relax\Widestg
\longest\ \the\result \\
\flong\ \the\result
```

results in:

```
     morning 21.64417pt
morning 21.64417pt
```

which is the same as that from `\Widesth`.

Although both macros give the same result I prefer Heiko's user interface to GL's, but then you may think it should be the other way round.

Tear along the dotted line.

Instruction, ANONYMOUS

2 Marching along

2.1 Oddment

On `ctt` Roger said that he was
. . . planning to take a string of the form mm.nn.pp where mm, nn, and pp are all integers, and test if pp is odd. So I'd like to write a macro that does that and use that as the parameter to `\ifodd`*.*

Joseph Wright responded [22] that it sounded as though he wanted something like:

```
\makeatletter
\newcommand*{\MyFunction}[1]{%
  \My@function#1..\@nil\@stop}
\def\My@function#1.#2.#3#4\@stop{
  \def\My@mm{#1}%
  \def\My@nn{#2}%
  \def\My@pp{}%
  \ifx#3\@nil\else
    \My@function@#3#4
  \fi
  % 0 below makes the test work when
  % \My@pp is empty
  \ifodd0\My@pp\relax
    Odd
  \else
    Even
  \fi}
\def\My@function@#1..\@nil{\def\My@pp{#1}}
\makeatother
...
\MyFunction{11.22.33}
\MyFunction{11.22.44}
\MyFunction{11.22.}
\MyFunction{11}
```

With Joseph's code, running his suggested test examples results in:

```
     Odd   Even   Even  Even
```

Quite frankly, I do not understand just how his code works. In order to get a better feel for it I decided to write my own macros for dot-separated lists of one, two, and three numbers and then try to extend them to deal with a list of arbitrary extent. Here are my efforts for the one, two, and three length lists. I included some diagnostic output to help when my code didn't work as I thought that it should.

Firstly, here are the code shorthands that I have used for the diagnostics — the `\cs` macro is defined in the ltugboat class, as shown.

```
%\DeclareRobustCommand\cs[1]{%
%  \texttt{\char`\\#1}}
\newcommand*{\sarg}[1]{\texttt{\{#1\}}}
\newcommand*{\csparg}[2]{\cs{#1}\sarg{#2}}
\newcommand*{\LRA}{%
  \ensuremath{\Longrightarrow} }
```

For a single number the command is:
`\MyFunctionI{`$\langle N\rangle$`}`

```
\newcommand*{\MyFunctionI}[1]{%
  \csparg{MyFunctionI}{#1} \LRA
  \ifodd0#1\relax
    #1 Odd
  \else
    #1 Even
  \fi}
```

Some example results are:

`\MyFunctionI{11}` $\Rightarrow$ 11 Odd
`\MyFunctionI{22}` $\Rightarrow$ 22 Even
`\MyFunctionI{}` $\Rightarrow$ Even

For a list of two numbers the command is:
`\MyFunctionII{`$\langle N.N\rangle$`}`

```
\makeatletter
\newcommand*{\MyFunctionII}[1]{%
  \csparg{MyFunctionII}{#1} \LRA
  \My@FunctionII#1\@nil
  \ifodd0\My@last\relax
    \My@last\ Odd
  \else
    \My@last\ Even
  \fi}
\def\My@FunctionII#1.#2\@nil{%
  \def\My@last{#2}}
\makeatother
```

Example results are:

`\MyFunctionII{11.22}` $\Rightarrow$ 22 Even
`\MyFunctionII{11.33}` $\Rightarrow$ 33 Odd
`\MyFunctionII{11.}` $\Rightarrow$ Even

For a list of three numbers the command is:
`\MyFunctionIII{`$\langle N.N.N\rangle$`}`

```
\makeatletter
\newcommand*{\MyFunctionIII}[1]{%
  \csparg{MyFunctionIII}{#1} \LRA
  \My@FunctionIII#1\@nil
  \ifodd0\My@last\relax
    \My@last\ Odd
  \else
```

```
    \My@last\ Even
  \fi}
\def\My@FunctionIII#1.#2.#3\@nil{%
  \def\My@last{#3}}
\makeatother
```

Some results are:

```
\MyFunctionIII{11.22.33} ⇒ 33 Odd
\MyFunctionIII{11.22.44} ⇒ 44 Even
\MyFunctionIII{11.33.} ⇒  Even
```

Based on the underlying idea — delimited arguments [1, 6, 9, 20] — of the above macros I then tried to develop one that would take a dot-separated list of any length and return whether the last number was odd or even.

I failed.

Eventually I remembered that the LaTeX kernel includes an `\@for` macro for marching along a comma-separated list of elements and decided to try and create a version that would handle dot-separated lists. It is effectively a copy of the `\@for` code replacing every '`,`' with a '`.`'. I can't pretend to understand how it works. I have named it `\@ford` as shorthand for '`\@fordot-separated-list`'.

```
\makeatletter
% \@ford NAME := LIST \do {BODY}
\long\def\@ford#1:=#2\do#3{%
  \expandafter\def\expandafter\@fortmp
  \expandafter{#2}%
  \ifx\@fortmp\@empty \else
    \expandafter
    \@forloopd#2.\@nil.\@nil\@@#1{#3}
  \fi}

\long\def\@forloopd#1.#2.#3\@@#4#5{%
  \def#4{#1}\ifx #4\@nnil \else
    #5\def#4{#2}\ifx #4\@nnil
    \else #5\@iforloopd #3\@@#4{#5}\fi\fi}

\long\def\@iforloopd#1.#2\@@#3#4{%
  \def#3{#1}\ifx #3\@nnil
  \expandafter\@fornoop \else
  #4\relax
  \expandafter\@iforloopd\fi#2\@@#3{#4}}
\makeatother
```

I did use this for a macro to handle unlimited length lists of the kind that Roger was interested in. Then there was a further posting from him [17] in response to Joseph (which I have abbreviated):

Thank you. That works (and was quite educational). However, I failed to completely specify my problem ...

Here's what I have:
`{a.b.c, x.y.z}` *or*
`{x.y.z}` *or*
`{, x.y.z}`
where `a,b,c, x,y,z` *are integers.*

What I would like to do is to be able to set a switch in the file that if set then the ... would be included only if `z` *is odd, but if the switch is not set then all ... will be included.*

This requirement seemed to me to be a candidate for a combination of `\@for` to handle the comma-separated parts and `\@ford` for the portions that are dot-separated.

Below is what I ended up with to handle an unlimited comma-separated list of unlimited dot-separated lists determining whether the last entry of all is odd or even.

First the `\DotFunction` for a dot-separated list of numbers. I have added some diagnostic print out just in case together with a means (`\ifop`) for enabling it. The macro is called like:
`\DotFunction{`$\langle N.N.N...N\rangle$`}`
and sets `\gotoddtrue` if the last number in the list is odd.

```
\newif\ifgotodd
\newif\ifop
\optrue

\makeatletter
\def\DotFunction#1{%
  \ifop \csparg{DotFunction}{#1} \LRA \fi
  \def\My@last{0}% in case arg is empty
  \@ford\scratch:=#1\do{%
    \edef\My@last{\scratch}}%
  \ifodd0\My@last\relax
    \gotoddtrue
    \ifop \My@last\ Odd \fi
  \else
    \gotoddfalse
    \ifop \My@last\ Even \fi
  \fi}
\makeatother
```

Some example results are:

```
\DotFunction{} ⇒ 0 Even
\DotFunction{11} ⇒  11 Odd
\DotFunction{11.22} ⇒  22 Even
\DotFunction{11.22.33} ⇒  33 Odd
\DotFunction{11.22.33.44} ⇒  44 Even
\DotFunction{11.nowt.33.44.55} ⇒  55 Odd
\DotFunction{11..33.44.55} ⇒  55 Odd
\newcommand*{\numM}{11.22.33.44.55.66.77}}
\DotFunction{\numM} ⇒
\DotFunction{11.22.33.44.55.66.77} ⇒ 77 Odd
```

Note that for `\DotFunction`, only the last element in the list must be an integer (or blank), earlier elements can be, for example, text. Further, unlike all the previous `\MyFunction...` macros, the argument may be a macro.

Finally, here is the end of the exercise — a generalised solution to Roger's requests, called as `\HisFuntion{`⟨*N.N...N, N...N, ..., N...N*⟩`}`

```
\makeatletter
\def\DotCommaFunction#1{%
\csparg{DotCommaFunction}{#1} \LRA
\opfalse% stop \DotFunction printing
  \@for\first:=#1\do{%
    \DotFunction{\first}}%
  \ifgotodd
\My@last\ Odd
  \else
\My@last\ Even
  \fi}
\makeatother
```

Some results of using `\DotCommaFunction`:

`\DotCommaFunction{1.2.3, 4.5.7}` ⇒ 7 Odd
`\DotCommaFunction{, 4.5.7}` ⇒ 7 Odd
`\DotCommaFunction{4.5.7}` ⇒ 7 Odd
`\DotCommaFunction{1, 2.3, , 4.5, 7.8}` ⇒ 8 Even
`\DotCommaFunction{1,2.3, ,4.5,7.8}` ⇒ 8 Even

All that remains is for the user to make appropriate changes to the actions of the odd/even result and to eliminate, or change, the diagnostic outputs to suit the application at hand.

2.2 Indexing into a list

Alastair asked [2]:
I've got a question about comma separated lists. Is there any way that you can index elements in a list. Lists can be iterated over in the PGF/TikZ package's `\foreach` *loop. How can you access an element whilst not in a loop?*

Often responses to questions on `ctt` provide the bare bones of a solution, leaving the questioner to adapt or extend it to his own situation. There were several responses and the one I found that would best suit me was from Ulrike Fischer [7]. The following is essentially Ulrike's code, edited to better fit the column:

```
\usepackage{tikz}
\def\values{i5, i4, i3, i2, i1}
\newcounter{loc}
\newcommand{\getitem}[1]{%
  \setcounter{loc}{0}
  \foreach \x in \values{%
    \stepcounter{loc}%
    \expandafter\xdef\csname
      alsval\the\value{loc}\endcsname{\x}}%
  \csname alsval#1\endcsname}
```

Ulrike's `\getitem{`⟨*N*⟩`}` macro returns the item that is in the ⟨*N*⟩th location in `\values` as `\alsvalN`. With:

```
\getitem{1}, \getitem{4}, \getitem{8}.
```

the result is:

i5, i2, .

I wondered if there was a solution that did not involve calling the `tikz` package and came up with the following which does not require any packages, being based on the LaTeX kernel's `\@for` construct.

```
\let\xpf\expandafter % just to save some space
\makeatletter
\newcount\vindex
\newcommand*{\getit}[2]{%
  \xpf\xpf\xpf\@getit\xpf{#2}{#1}%
  \theans}

\newcommand*{\@getit}[2]{%
\vindex=0
\def\theans{Index #2 is out of range.}%
  \xdef\alist{#1}%
  \@for\tmp := #1 \do{%
    \advance\vindex 1
    \ifnum\the\vindex=#2
      \xdef\theans{\tmp}%
    \fi}}
\makeatother
```

The macro `\getit{`⟨*N*⟩`}{`⟨*list*⟩`}` returns `\theans` as the value of the ⟨*N*⟩th item in the ⟨*list*⟩ where ⟨*list*⟩ may be either a comma-separated list or a macro defined as one. I have included a check on whether ⟨*N*⟩ is valid for the given list (this would be better in the form of an error report in the `log` file external to the document instead of being typeset).

With these inputs

```
\getit{1}{\values},
\getit{4}{\values},
\getit{8}{\values}.

\getit{1}{i5, i4, i3, i2, i1},
\getit{4}{i5, i4, i3, i2, i1},
\getit{8}{i5, i4, i3, i2, i1}.
```

the results are:

i5, i2, Index 8 is out of range.
i5, i2, Index 8 is out of range.

The key problem that I had to solve in my method is that the 'list' that `\@for` operates on must be an actual sequence of comma-separated items and not a macro defined as such a list. That is why I have separated the code into two macros. The first to grab the list, be it actual or as a macro, and then to hand that over to `\@getit` as an actual list by utilising a series of `\expandafter`s within `\getit`.[1]

The tumult and the shouting dies,
The captains and the kings depart,
And we are left with large supplies
Of cold blancmange and rhubarb tart.

After the Party, Ronald Knox

3 A blank argument

The title of a posting by Matthew to `texhax` was *Finding blank argument to a macro.* There is a long history behind this kind of macro, initially posed as a challenge in Michael Downes' *Around the Bend* [5] series in the early 90s, and without looking any further I assumed that the solution would be the ifmtarg [3] package which provides a test as to whether a macro argument consists of zero or more blank spaces.

However, I was mistaken, as Matthew's posting continued [11]:

I am trying to solve a problem in LaTeX that I thought would be relatively straightforward. I would like to make a macro that will evaluate its argument and tell me whether the result is blank or not . . . I managed to come up with a TeX macro that handles different types of 'blank' pretty well. It properly recognizes an empty argument, empty braces, spaces, etc. It even works on another macro that evaluates to a blank, so I thought I was home free. However, as soon as I fed it a macro that takes an argument, bad things happen. I've attached a simple document below that shows the problem.

The 'simple document' contains many lines of code implementing his `\blankArgTest{`⟨*arg*⟩`}`, together with examples of when it worked and when it didn't give the required result. With the macros:

```
\usepackage{ifthen}
\newcommand{\testA}{%
  \ifnum10=10 \empty\else A\fi}
\newcommand{\testB}{%
  \ifthenelse{10=10}{\empty}{B}}
\newcommand{\testC}[1]{%
  \ifnum#1=10 \empty\else C\fi}
```

`\blankArgTest` worked when ⟨*arg*⟩ was `\testA` but failed for `\testB` and `\testC`.

Michael Barr [4] came up with a remarkably simple solution which I am presenting as:

```
\newcommand{\IfBlank}[1]{%
  \setbox0\hbox{$#1$}%
  \ifdim\wd0=0pt
    Blank
  \else
    Not blank
  \fi}
```

The basic idea is to put the argument into an `\hbox` and check if the box's width is zero. This assumes that a 'blank' argument is one that results in no typeset material (or rather, anything typeset ends with zero width). With the following definitions:

```
\newcommand{\blank}{ }
\newcommand{\tout}{\typeout{Typeout}}
```

examples of the `\IfBlank` macro are:

```
\IfBlank{} Blank
\IfBlank{    } Blank
\IfBlank{Text} Not blank
\IfBlank{\blank} Blank
\IfBlank{\tout} Blank
\IfBlank{{ }} Blank
\testA Blank
\testB Blank
\testC{10} Blank
```

A somewhat different need for an empty/blank argument was expressed by Timothy Murphy who wrote [13]:

I have a macro `\cmd#1#2` *. Both arguments are given in the form* `{...}`*. I'd like an empty second argument* `{}` *to be added if none is given, i.e., if the next character after* `\cmd{...}` *is not* `{` *.*

What is the simplest way to do this?

There were three interesting proposed solutions which I have given below.[2]

Heiko Oberdiek's was the first positive response and was essentially as follows [15]:

```
\newcommand*{\CmdH}[1]{%
  \begingroup
  % remember parameter
  \toks0={#1}%
  % look forward
  \futurelet\NextToken\CmdI}
\newcommand*{\CmdI}{%
  \ifx\NextToken\bgroup
    \edef\next{\endgroup
      \noexpand\CmdImpl{\the\toks0}}%
  \else
    \edef\next{\endgroup
      \noexpand\CmdImpl{\the\toks0}{}}%
  \fi
```

[1] `\expandafter` and when it should be used is to me among the more difficult aspects of TeX code. I usually come to a solution by either following what others have done in similar circumstances or by much experimentation — otherwise known as errors and trials.

[2] I have slightly edited the code, principally by using distinguished macro names instead of the somewhat generic `\cmd`, and using a common set of tests.

```
  \next}
\newcommand{\CmdImpl}[2]{%
  \textbf{Heiko:}
  this is (#1) and here’s (#2).}
\CmdH{abc}{def} \\
\CmdH{ghi}\relax \\
\CmdH{jkl} %
```

which results in:

Heiko: this is (abc) and here’s (def).
Heiko: this is (ghi) and here’s ().
Heiko: this is (jkl) and here’s ().

Dan Luecking [10], noting *that there were probably better ways (see Heiko’s reply)*, responded with:

```
\makeatletter
\newcommand*{\CmdD}[1]{%
  \@ifnextchar\bgroup
    {\Cmd@i{#1}}{\Cmd@i{#1}{}}}
\newcommand*{\Cmd@i}[2]{%
  \textbf{Dan:}
  this is (#1) and here’s (#2).}
\makeatother
\CmdD{abc}{def} \\
\CmdD{ghi}\relax \\
\CmdD{jkl} %
```

which results in:

Dan: this is (abc) and here’s (def).
Dan: this is (ghi) and here’s ().
Dan: this is (jkl) and here’s ().

Dan pointed out that his code does not examine the actual *next* character, but rather the next *nonspace* character. He also commented that Heiko’s solution emulates a portion of `\@ifnextchar` without the skipping of spaces.

Joseph Wright [21] proposed a solution based on the xparse package [19] developed as part of the LaTeX3 project. From the user’s viewpoint it appears to be the simplest of the three proposed solutions.

```
\usepackage{xparse}
\NewDocumentCommand\CmdJ{mG{}}{%
  \textbf{Joseph:}
  this is (#1) and here’s (#2).}
\CmdJ{abc}{def} \\
\CmdJ{ghi}\relax \\
\CmdJ{jkl} %
```

which results in:

Joseph: this is (abc) and here’s (def).
Joseph: this is (ghi) and here’s ().
Joseph: this is (jkl) and here’s ().

The three very different implementations each handled all the test cases correctly.

America is a land whose center is nowhere;
England one whose center is everywhere.

Pick Up Pieces, John Updike

4 A centered table of contents

Bogdan Butnaru[3] uses the memoir class and asked me how to have a centered table of contents (ToC). I came up with one solution and passed Bogdan’s request on to Lars Madsen, who is now memoir’s maintainer, and he came up with a better solution; both of these were based on memoir’s tools for manipulating the ToC. I then came up with a more basic solution which is also applicable to the standard book and report classes.

In these classes a chapter entry is set by the `\l@chapter` macro, a section entry by `\l@section`, and so on. These may be redefined to produce centered entries. These macros have the general calling form of:
`\l@chapter{`⟨*number-and-title*⟩`}{`⟨*page*⟩`}`
where ⟨*number-and-title*⟩ has the form:
`{{\numberline}{num} title}`
where `\numberline` typesets the chapter number. The `\l@...` macros also take into account whether or not the entry should be printed and the surrounding vertical spacing. The *LaTeX Companion* [12, §2.3] provides further information about ToCs and related packages.

The following redefinition of `\l@chapter` will center the chapter entries, with the chapter number above the title, and a middle-dot between the title and page number.

```
\makeatletter
\renewcommand*{\l@chapter}[2]{%
  \ifnum\c@tocdepth>\m@ne % print chapter entry
  \addpenalty{-\@highpenalty}%
  \vskip 1em plus 0pt
  \begingroup
    \def\numberline##1{##1\\\nobreak}% number
    {\centering\bfseries
      #1~\textperiodcentered~#2\par}%
  \endgroup
  \fi}
\makeatother
```

The `\tableofcontents` macro uses `\chapter*` to set the title ragged right. A hack to that can be used to center the title is to make `\raggedright` into `\centering`.

```
\let\saverr\raggedright
```

[3] Private email, 2010/07/21

```
\newcommand*{\rrtocenter}{%
  \let\raggedright\centering}
\newcommand*{\restorerr}{%
  \let\raggedright\saverr}
\let\oldtoc\tableofcontents
\renewcommand*{\tableofcontents}{%
  {\rrtocenter\oldtoc}}
```

To typeset the ToC with the heading and chapter entries centered is now as easy as:

```
\tableofcontents
```

If you wanted the section entries to be centered then `\l@section` can be redefined in a similar, but not identical, manner to `\l@chapter`. However, centered section entries following a centered chapter entry in my view looks rather confusing.

If you want chapter headings to be centered, you can do:

```
{\rrtocenter
   \chapter[...]{...}
}
```

or

```
\rrtocenter
   \chapter[...]{...}
\restorerr
```

In each case the effect of `\rrtocenter` is limited to `\chapter`; if it were not then surprises could be in store later on.

References

[1] Paul W. Abrahams, Karl Berry, and Kathryn A. Hargreaves. *TEX for the Impatient*. Addison-Wesley, 1990. `http://ctan.org/pkg/impatient`.

[2] Alastair. Indexing individual elements in a comma separated list. Post to `comp.text.tex` newsgroup, 17 October 2010.

[3] Donald Arseneau and Peter Wilson. The ifmtarg package, 2009. `http://ctan.org/pkg/ifmtarg`.

[4] Michael Barr. Re: [texhax] Finding blank argument to a macro. Post to `texhax` mailing list, 27 May 2010.

[5] Michael Downes (ed. Peter Wilson). *Around the Bend*. The Herries Press, July 2008. `http://ctan.org/pkg/around-the-bend`.

[6] Victor Eijkhout. *TEX by Topic, A TEXnician's Reference*. Addison-Wesley, 1991. ISBN 0-201-56882-9. Available at `http://www.eijkhout.net/tbt/`.

[7] Ulrike Fischer. Re: Indexing individual elements in a comma separated list. Post to `comp.text.tex` newsgroup, 18 October 2010.

[8] GL. Re: Finding the widest string. Post to `comp.text.tex` newsgroup, 4 May 2010.

[9] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984. ISBN 0-201-13448-9.

[10] Dan Luecking. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.

[11] Matthew. [texhax] finding blank argument to a macro. Post to `texhax` mailing list, 25 May 2010.

[12] Frank Mittelbach and Michel Goossens. *The LATEX Companion*. Addison Wesley, second edition, 2004. ISBN 0-201-36299-6.

[13] Timothy Murphy. A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.

[14] Heiko Oberdiek. The kvsetkeys package, 2010. `http://ctan.org/pkg/kvsetkeys`.

[15] Heiko Oberdiek. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.

[16] Heiko Oberdiek. Re: Finding the widest string. Post to `comp.text.tex` newsgroup, 4 May 2010.

[17] Roger. Re: ifodd question. Post to `comp.text.tex` newsgroup, 18 May 2010.

[18] Romildo. Finding the widest string. Post to `comp.text.tex` newsgroup, 3 May 2010.

[19] The LaTeX3 Project. The xparse package, 2015. `http://ctan.org/pkg/xparse`.

[20] Peter Wilson. Glisterings: More on paragraphs regular, LATEX's defining triumvirate, TEX's dictator. *TUGboat*, 29(2):324–327, 2008. `http://tug.org/TUGboat/tb29-2/tb92glister.pdf`.

[21] Joseph Wright. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.

[22] Joseph Wright. Re: ifodd question. Post to `comp.text.tex` newsgroup, 14 May 2010.

◇ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
`herries dot press (at) earthlink dot net`