Hints & Tricks

Glisterings

Peter Wilson

> Not all that tempts your wand'ring eyes
> And heedless hearts, is lawful prize;
> Nor all, that glisters, gold.
>
> *Ode to a Favourite Cat*
> Thomas Gray

For many years Jeremy Gibbons has edited a very successful column in *TEX and TUG NEWS* and *TUGboat* called *Hey — It works!*[**?**]. I have learnt much from this but apparently not enough to decline when asked to take over the column. On the other hand I have learnt to my cost that the quickest way to get a correct answer to a question on the `comp.text.tex` (`ctt`) newsgroup is to give an incorrect answer. In order not to sully Jeremy's reputation my first thought was to change the title to *Hey — It might work* but after some consideration the new title is as you see it above — *Glisterings* — implying that there might be some dross among the nuggets.

Corrections, suggestions, and contributions will always be welcome.

Several questions on `ctt` recently have been related to comparing two words or strings. To my chagrin I gave an incorrect answer to one of the questions, so I'll now try and redeem myself.

> If you can meet with triumph and disaster
> And treat those two imposters just the same...
>
> *If —*
> Rudyard Kipling

Checking for an optional argument

If you are defining a new command that has an optional argument you often need some way of checking whether or not it is present when the macro is called, especially when it should be ignored if it is not present. One convention is to use the kernel `\@empty` macro as the default for the optional argument.

```
\newcommand{\mine}[2][\@empty]{%
%  if #1 is \@empty do nothing else
%  do something
```

To me the obvious way of performing the check was to use TEX's `\ifx` primitive to compare `\@empty` and the actual value of the argument, as in

```
\newcommand{\testoptarg}[1][\@empty]{%
  \ifx #1\@empty
    Optional (#1) unused%
  \else
    Optional (#1) present%
\fi}
```

If you try this you can get some odd results:
`\testoptarg` Optional () unused
`\testoptarg[full]` Optional (full) present
`\testoptarg[oops]` psOptional (oops) unused

It was kindly explained to me[1] that `\ifx` checks the following two tokens and in TEX a token is either a command sequence (e.g., `\@empty`) or a single character, like 'o'. In the oops example, `\ifx` checks 'o' and 'o', concludes that they are the same, and hence the strange result. Flipping the token ordering works better:

```
\ifx\@empty#1
```

Now `\testoptarg` and `\testoptarg[\@empty]` will report 'Optional () unused'. Any other call, for example `\testoptarg[]`, will report 'Optional () present', and in particular `\testoptarg[oops]` reports 'Optional (oops) present'.

String comparisons

A more general problem along the same lines is to check if two words, or strings are the same. We can use `\ifx` for this as well. When `\ifx` compares two tokens that are macro names, the result is true if the macros have been defined in the same way, and if their first level replacement texts are the same. So, we define two macros whose replacement texts are the strings, and compare these.

```
\newif\ifsame
\newcommand{\strcfstr}[2]{%
  \samefalse
  \begingroup
    \def\1{#1}\def\2{#2}%
    \ifx\1\2\endgroup \sametrue
    \else   \endgroup
    \fi}
```

The two arguments to `\strcfstr`[2] are the strings to be tested. `\ifsame` is set true if the two strings match character to character. If the arguments are macro names it checks the characters in the names, not their definitions. If there are any spaces in the arguments, each group is reduced to a single space before the strings are compared. `\strcfstr{}{␣}` sets `\ifsame` false but `\strcfstr{␣}{␣␣}` sets it true.

```
\newcommand{\StrCfStr}[2]{%
  \lowercase{\strcfstr{#1}{#2}}}
```

The `\StrCfStr` macro performs a case insensitive test on two strings. For example, it will set `\ifsame` true for any of the pairs (abc, abc), (abc, Abc), (abc, aBc), and so on. It uses `\lowercase` to convert any uppercase letter to a lowercase letter so all the letters will be lowercase at the time `\strcfstr` does the checking. This will not work if the arguments include differently cased macro names as `\lowercase` does not touch those.

The `\strcfstr` and `\StrCfStr` macros have provided all the string testing that I have needed, but I'll show a couple of extensions. One thing is that `\strcfstr` relies on `\def` which is not expandable so, for example, it cannot be used in an `\edef`. Both Victor Eijkhout [**?**, section 13.8.7] and David Kastrup [**?**] have presented solutions for this. The other is that you may want to check if a macro expands to a particular string. David's expandable macro also provides a solution for this and Michael Downes[**?**] gives a somewhat different method using `\expandafter`.

We can use `\strcfstr` as the basis for the macro to string comparison, by using `\expandafter`s.

```
\newcommand{\macrocfstr}[2]{%
  \expandafter\strcfstr\expandafter{#1}{#2}}
```

The first argument to `\macrocfstr` is either a string or a macro that is expected to expand to a string. The second argument is the test string.

We can also do a case insensitive test by using

```
\newcommand{\MacroCfStr}[2]{%
  \lowercase{\macrocfstr{#1}{#2}}}
```

The `\charscfchars` expandable macro below is based on Victor's code. It is tricky because it uses recursion to perform pairwise comparisons of the individual characters in its two arguments, and it requires two supporting macros.

```
\catcode`\^^G=11   % make a letter
\newcommand{\charscfchars}[2]{%
  \IfAllChars#1^^G\Are#2^^G\theSame}
```

`\charscfchars` adds a character at the end of its arguments to mark the ends of the strings. Victor used $ as the marker which meant that neither argument could include $ among the characters. I chose to use `^^G` (TEX's notation for the ASCII BEL control character, which is normally invalid TEX). The `\catcode` changes first make `^^G` appear to be a letter and then at the end of the macro definitions it is set back to its normal invalid state.

The next macro, which is presented with some interspersed commentary, does most of the work.

```
\def\IfAllChars#1#2\Are#3#4\TheSame{%
  \if#1^^G\if#3^^G\sametrue
          \else\samefalse\fi
```

[1] By, among others, Donald Arseneau, Michael Downes and Stephan Lemke.

[2] The `cf` used in the names of macros is the abbreviation *cf* (from the Latin *confer* = compare).

The macro takes two pairs of arguments that are delimited by the tokens `\Are` and `\TheSame`. The first pair of arguments are for the first string under test and the second pair for the other string. More specifically, `#1` will be the first character in the first string and `#2` contains the remaining characters (including the `^^G` marker), and similarly for `#3` and `#4`. If the ends of both strings have been reached, then the strings are the same, but if only the end of the first string has been reached, the strings are different. If we are not at the end of the first string there is more work to be done.

```
  \else\if#1#3\IfRest#2\TheSame#4\else
       \samefalse\fi\fi}
```

If the corresponding characters in the two strings are the same then the rest of the character pairs must be checked, otherwise the characters don't match and we are done.

The last of the macros takes three arguments which are delimited by the tokens `\TheSame`, `\else`, and `\fi\fi`. The first two arguments are strings to be compared, and it throws away the third.

```
\def\IfRest#1\TheSame#2\else#3\fi\fi{%
  \fi\fi \IfAllChars#1\Are#2\TheSame}
\catcode`\^^G=15  % return to invalid
```

This macro simply calls `\IfAllChars...` to compare the strings.

`\charscfchars` can be used as a basis for case insensitive and macro to string comparisons exactly like `\strcfstr`.

Apart from `\charscfchars` being expandable while `\strcfstr` is not, it also ignores all space characters while `\strcfstr` does not. For example, `\charscfchars{ab}{a␣b}` thinks that the arguments are identical but they will be reported as different if `\strcfstr{ab}{a␣b}` is used.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166 USA
herries.press@earthlink.net