The Macroswap Package*

Robert J Lee
latex@rjlee.homelinux.org

August 18, 2013

Abstract

This package provides macros to allow the user to redefine a pair of macros so that the token lists they expand to will be swapped.

1 Introduction

This package was written to allaw a simple, readable syntax for swapping the values within macros, used as program variables.

Usage

`\macroswap`

The `\macroswap` macro allows you to temporarily swap two macro definitions; for example:

```
\newcommand{\myfirst}{1}
\newcommand{\myend}{2}
\begingroup
 \macroswap{myfirst}{myend}
 \myfirst\myend % prints ``21''
\endgroup
\myfirst\myend % prints ``12'' (swapped defininions were local)
```

This local swap is often unwanted, and can use up TeX's stack memory. So you may prefer the global version:

`\gmacroswap`

The `\gmacroswap` macro allows you to globally swap two macro definitions; for example:

```
\newcommand{\myfirst}{1}
\newcommand{\myend}{2}
```

*This document corresponds to Macroswap ?, dated ?.

```
\begingroup
 \gmacroswap{myfirst}{myend}
 \myfirst\myend % prints ``21''
\endgroup
\myfirst\myend % prints ``21'' (swapped defininions are global)
```

Usage with arrayjobx

To globally swap two elements of an array declared with the `arrayjobx` package, you need to arrange to swap the macros used by that package. This is not part of the public interface and may change at any time; unfortunately, it does not seem possible to do this with only the public interface, especially if you do not want to expand the tokens in the array.

Assuming arrayjobx version 1.04 (labelled "05/03/2010"), it is only necessary to swap the values of the underlying macros using the following syntax:

```
\newarray{arr}
\readarray{arr}{A&C&B}
\gmacroswap{arr2\string~}{arr3\string~}
\arr(1)\par % prints ``A''
\arr(2)\par % prints ``B''
\arr(3)\par % prints ``C''
```

Implementation

First we make the @ symbol a letter so wo can use it internally

```
1 \makeatletter
```

`\gmacroswap` given the name of two macros, swap their definitions. Produces no output. This version is global.

```
2 \newcommand{\gmacroswap}[2]{%
```

NB: we don't use ifcsname, because there's no guarantee that `#1` or `#2` will be a single token `macro@swap@left` and `macro@swap@right` are the contents of the passed-in macro names. The `edef` would normally fully expand its parameter but the `csexpandonce` prevents the full expansion and returns only the token list that we want.

```
3   \edef\macro@swap@left{\csexpandonce{#1}}%
4   \edef\macro@swap@right{\csexpandonce{#2}}%
```

Having taken a copy of the passed-in macro, we then reassign it to the values in our tokens, which we get using `expandonce` to strip off our temporary macro names. Again, `xdef` would normally expand its macros as well as being a global assignment; however, the `expandonce` prevents the full expansion.

`expandafter` is used to evaluate the `csname`...`\endcsname` become the `xdef`, as `xdef` will only redefine its first parameter.

```
  \expandafter\xdef\csname#2\endcsname{\expandonce{\macro@swap@left}}%
  \expandafter\xdef\csname#1\endcsname{\expandonce{\macro@swap@right}}%
```

Finally, we throw away our temporary macros:

```
  \let\macroswap@left\relax%
  \let\macroswap@right\relax%
}
```

`\macroswap` given the name of two macros, swap their definitions. Produces no output. This version is local.

```
\newcommand{\macroswap}[2]{%
```

NB: we don't use ifcsname, because there's no guarantee that `#1` or `#2` will be a single token

```
  \edef\macro@swap@left{\csexpandonce{#1}}%
  \edef\macro@swap@right{\csexpandonce{#2}}%
```

The only difference here is the use of `edef` instead of `xdef`, which expands its second argument and overwrites its first argument locally, so the original values will be restored at the end of the current group or environment.

```
  \expandafter\edef\csname#2\endcsname{\expandonce{\macro@swap@left}}%
  \expandafter\edef\csname#1\endcsname{\expandonce{\macro@swap@right}}%
  \let\macroswap@left\relax%
  \let\macroswap@right\relax%
}
```

Leave @ as another character

```
\makeatother
```

That is all

Change History

v1.0

General: Initial version 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.