Showing-off math macros

Michael J. Wichura

Introduction

The chore of typing math formulas is made less onerous by using macros to `\define` simple abbreviations (such as `\xvec`) for complicated constructions (such as $(x_1, \ldots, x_n)$) that occur repeatedly in a document. This saves keystrokes and cuts down on typographical errors and inconsistencies. The only difficulty is remembering the abbreviations and what they stand for. This article presents a macro-listing command that is helpful in this regard. The command typesets a file containing math macro `\definitions`, such as

```
\def\xvec{\row x}
\def\row#1{\Row{#1}n}
\def\Row#1#2{(#1_1,\ldots,#1_{#2})}
\def\ex{\e x}
\def\e#1{e^{-#1^2\2}}
\def\tenth{\fr 1/10 }
\def\fr#1/#2 {\textstyle{#1\over #2}}
\def\eunm{{n \euler m}}
\def\euler{\atopwithdelims<>}
\def\tnk{\take k \of n}
\def\take#1\of#2{{}_{#2}C_{#1}}
```

in the form of a table, like this:

Macros on file: `macros.math`

Name and Parameter Text	*Value*	*Replacement Text*
`\xvec`	$(x_1, \ldots, x_n)$	`\row x`
`\row#1`	$(\#1_1, \ldots, \#1_n)$	`\Row {#1}n`
`\Row#1#2`	$(\#1_1, \ldots, \#1_{\#2})$	`(#1_1,\ldots ,#1_{#2})`
`\ex`	$e^{-x^2/2}$	`\e x`
`\e#1`	$e^{-\#1^2/2}$	`e^{-#1^2\!/2}`
`\tenth`	$\frac{1}{10}$	`\fr 1/10`
`\fr#1/#2␣`	$\frac{\#1}{\#2}$	`\textstyle {#1\over #2}`
`\eunm`	$\left\langle {n \atop m} \right\rangle$	`{n \euler m}`
`\euler`	$\langle\rangle$	`\atopwithdelims <>`
`\tnk`	${}_nC_k$	`\take k \of n`
`\take#1\of␣#2`	${}_{\#2}C_{\#1}$	`{}_{#2}C_{#1}`

(`macros.math` is the name of the file containing the `\definitions` used in this example.) The table is more informative than a simple print-out because it shows what each macro does. You can easily locate the macro that produces a certain construction by scanning down the middle column.

The macro-listing command is

`\ListMacrosOnFile` *file_name*␣

where *file_name* is the name of a file containing `\definitions` of math-mode macros, and only such `\defs`. This file would be one that you'd ordinarily `\input` as part of a document: no special organization is required. Several definitions can be given on a single line. A single definition can extend over several lines. Blank lines are allowed, as are `\input` commands referring to files containing more math-mode macros.

The table that `\ListMacrosOnFile` produces contains a row for each `\def` on *file_name*. Recall that the syntax for a macro definition is

`\def`⟨*control sequence name*⟩⟨*parameter text*⟩ {⟨*replacement text*⟩}

(see page 203 of *The TEXbook*). `\ListMacrosOnFile` places the macro's name and parameter text in the first column of the table, with spaces in the parameter text being shown as '␣'. It's important to take note of those spaces, because they can play a role in delimiting arguments to the macro; see pages 203–204 of *The TEXbook*. The second column shows the effect the macro has when it is typeset between a pair of `$` signs and invoked with arguments `{\rm \#1}`, `{\rm \#2}`, … corresponding to the parameters `#1`, `#2`, … in an obvious manner. For example, the "value" of the `\take` macro in the opening example was created by the construction '`$\take {{\rm \#1}}\of {{\rm \#2}}$`'. Finally, the macro's replacement text is given in the third column. An overwide entry in the first or second column is allowed to stick out to the right and causes its row to be continued on the next line. A long entry in the third column is carried over to subsequent lines.

There are two significant limitations on the use of `\ListMacrosOnFile`. The first has already been mentioned but is worth repeating: *file_name* and its `\input` extensions must contain only `\def`initions of macros that can be used in math mode.* For example, '`\def\AB{\alpha + \beta}`' is allowed, but '`\def\AB{$\alpha$ and $\beta$}`' is not, since in this case the construction '`$\AB$`' would elicit an error message from TeX. The second limitation is that the expansion of the replacement text for each macro must be closed with respect to groups. For example, '`\def\BeginBox{\setbox0 = \hbox\bgroup}`' is not allowed because the group opened by `\bgroup` isn't closed by a matching `\egroup`.

`\ListMacrosOnFile` will correctly handle any parameter text TeX allows, with two minor exceptions: that text must not contain the construction '`->`', nor end with the character '`#`'.

*It would be nice if there were a simple, automatic way to show what a formatting macro like plain TeX's `\beginsection` command does. Unfortunately, such macros typically have a wide-ranging effect that can't be encapsulated in a table entry.

The macros

`\ListMacrosOnFile` is implemented through a collection of interrelated macros. I'll first discuss the ones that deal with alignment. `\ListMacrosOnFile` doesn't use an `\halign` to construct its table because extremely wide entries in the first or second column could push the entire third column off the page, and because a long list of `\def`s on *file_name* could produce a multipage table exceeding TeX's memory. An alignment mechanism that uses preset column widths avoids both of these problems because it allows each row to be typeset independently of the other rows. Plain TeX's `\+` command functions in this way, but lets an overwide entry in one column overlap the following one, producing illegible effects likthis. `\ListMacrosOnFile` employs an "in house" alignment in which each row is individually set using an `\halign`, the templates of which position entries flush left in columns of preset widths and avoid overlaps by inserting a `\cr` and an appropriate number of `&`'s after an overwide entry.

The `\SetColumnWidths` command defined by

```
\def\SetColumnWidths#1#2#3{%
  % #1, #2, and #3 are dimensions
  \def\NameColumnWidth{#1}%
  \def\ValueColumnWidth{#2}%
  \def\ReplacementColumnWidth{#3}}
```

is used to set the three column widths to user-defined values. The macros specify

```
\SetColumnWidths
  {.25\hsize}{.20\hsize}{.55\hsize}
```

to establish defaults that experience has shown to work out fairly well. The alignment macros are `\BeginAlignRow` and `\EndAlignRow`:

```
\def\BeginAlignRow{%
  \xdef\AmpsSeenSoFar{}%
  \ialign \bgroup
    \BeginColumn ##\EndColumnOfWidth\NameColumnWidth
    &\BeginColumn ##\EndColumnOfWidth\ValueColumnWidth
    &\BeginColumn ##\EndColumnOfWidth\ReplacementColumnWidth
    \cr}
\def\EndAlignRow{\egroup}
```

and their subsidiaries `\BeginColumn` and `\EndColumnOfWidth`:

```
\def\BeginColumn{\setbox0 = \hbox \bgroup}
\def\EndColumnOfWidth#1{%
  \ifLastColumn  \egroup % now box0 holds the entry
      \box0
      \else
          \egroup % now box0 holds the entry
          \setbox2 = \hbox to #1{\unhcopy0 \hss}%
          \copy2
          \xdef\AmpsSeenSoFar{\AmpsSeenSoFar &}%
          \ifdim\wd0 > \wd2
            \xdef\DropDownToNextLine{%
              \noexpand\LastColumntrue\cr
              \noalign{\noexpand\nobreak}%
              \AmpsSeenSoFar}%
            \xdef\AmpsSeenSoFar{}%
            \aftergroup\DropDownToNextLine
          \fi
      \fi}
    \newif\ifLastColumn  % false by default
```

The `\ifdim` clause at the end of `\EndColumn`... is what prevents overlaps. To see what's involved, suppose the `\ifdim` test has just discovered that an entry in the second column is overwide (`\wd0` > `\wd2`). Then the `\aftergroup` command will effectively insert the tokens `\LastColumntrue \cr \noalign{\nobreak} &&` in front of the input text for the third column, causing TEX to finish off the current line with an empty third column, issue a penalty preventing a page break, and start a new line with two empty columns, before going on to set the next entry. `\ListMacrosOnFile` specifies `\LastColumntrue` when it's working on the third column, so that that column is never considered to be overwide.

`\ListMacrosOnFile` itself comes next. It first `\inputs` *file_name*, so that all the macros there will be defined, and creates the header lines for the table. It then `\inputs` *file_name* once again, but with the meaning of `\def` changed to `\BeginExhibitMacro`. `\def`'s normal meaning is restored after all the macros on *file_name* have been exhibited:

```
    \def\ListMacrosOnFile #1 {%
      \par \rm
      \input #1
      \leftline {Macros on file: {\tt #1}}
      \medskip
      \leftline {\it Name and}
      \BeginAlignRow
        \it Parameter Text\quad &\it Value\quad  &\it Replacement Text\cr
        \EndAlignRow
      \ListMacros
      \input #1
      \DontListMacros}
    \def\ListMacros{\let\def = \BeginExhibitMacro}
    \def\DontListMacros{\let\def = \PrimitiveDef}
    \let\PrimitiveDef = \def
```

`\ExhibitMacro`'s job is to construct a row displaying a macro's name and parameter text (e.g., `\take#1\of␣#2`), value (e.g., ${}_{\#2}C_{\#1}$), and replacement text (e.g., `{}_{#2}C_{#1}`). Since the name and texts immediately follow the `\def` on *file_name*, there would at first sight seem to be no difficulty in fabricating the entries for the first and third columns. It turns out, though, that the parameter text on *file_name* has to be used to create the value entry in the second column, and that (due to category code considerations) this precludes the use of that same text in the first column. The way out of this bind is to ask TEX for the `\meaning` of ⟨*name*⟩ and to extract the relevant texts from TeX's response

`macro:`⟨*parameter text*⟩`->`⟨*replacement text*⟩

All the non-space characters in TeX's response have category code 12, so the extracted texts can be typeset without using a verbatim command:

```
\def\BeginExhibitMacro#1{% #1 = macro's name
  \DontListMacros
  \smallskip
  \BeginAlignRow
    \gdef\Name{#1}% save name for later use
    \expandafter\ExtractTexts\meaning#1\EndExtractTexts
    \tt \frenchspacing
    \expandafter\string\Name
    \expandafter\MakeSpacesVisibleA \ParameterText\EndMakeSpacesVisible
    \quad
  &
    \vtop\bgroup\Argument={}\BuildArgumentA
  }
\def\EndExhibitMacro#1{% #1 = macro's replacement text
      \ialign {\span\the\ValueTemplate\cr
        \expandafter\Name\the\Argument\cr}
      \egroup
    \quad
  &
    \vtop {%
      \hsize = \ReplacementColumnWidth \tt \frenchspacing
      \noindent \hangindent=1em \rightskip=0pt plus 4em \relax
      \ReplacementText}%
    \LastColumntrue
  \cr
  \EndAlignRow
  \ListMacros}
\def\ExtractTexts #1:#2->#3\EndExtractTexts{%
  \gdef\ParameterText{#2}%
  \gdef\ReplacementText{#3}}
```

The replacement text is set ragged right in a paragraph having the width of the third column. The following macros are used in converting the (invisible) spaces in the parameter text to visible characters (␣'s) before that text is placed in the first column:

```
\def\\{\let\SpaceToken= }\\
\def\SpaceChar{\char`\ }
\def\MakeSpacesVisibleA{\futurelet\NextToken\MakeSpacesVisibleB}
\def\MakeSpacesVisibleB{\ifx \NextToken\SpaceToken \SpaceChar \fi
  \MakeSpacesVisibleC}
\def\MakeSpacesVisibleC#1{%
  \ifx #1\EndMakeSpacesVisible \else #1\expandafter\MakeSpacesVisibleA \fi}
\def\EndMakeSpacesVisible{}
```

`\ExhibitMacro`'s code for the second column starts out by fabricating an "argument text" (e.g., `{{\rm \#1}}\of {{\rm \#2}}`) out the parameter text on *file_name*. This operation is a little delicate, because category codes have to be assigned correctly, TeX's rules regarding spaces have to be obeyed, and active characters and control sequences (like the `\of` in the `\take` macro) mustn't be expanded. The following macros are used in building up the argument text token by token.

```
\def\BuildArgumentA{\futurelet\NextToken\BuildArgumentB}
\def\BuildArgumentB{%
  \ifx \NextToken\bgroup
    \def\NextCmd{\EndExhibitMacro}%
  \else \ifx ##\NextToken
    \def\NextCmd{\AddParmFieldToArgument}%
  \else \ifx \NextToken\SpaceToken
    \def\NextCmd{\AddSpaceToArgument}%
  \else  \def\NextCmd{\AddOtherToArgument}%
  \fi \fi \fi
  \NextCmd}
\newtoks\Argument
\def\AddToArgument#1{\edef\temp{\the\Argument#1}%
  \Argument=\expandafter{\temp}%
  \BuildArgumentA}
\def\AddParmFieldToArgument#1#2{%
  \AddToArgument{{{\noexpand\rm \noexpand\##2}}}}
\def\AddSpaceToArgument#1 {\AddToArgument\space}
\def\AddOtherToArgument#1{\AddToArgument{\noexpand #1}}
```

The value entry for the second column is created by typesetting the construction '⟨*macro name*⟩ ⟨*argument text*⟩' (e.g.,

```
\take {{\rm \#1}}\of {{\rm \#2}}
```

) in a one row, one column `\halign` whose template is `$##$` by default:

```
\newtoks\ValueTemplate
\def\ShowOffMathMacros{%
  \ValueTemplate={$##$}}
\ShowOffMathMacros
```

There are, however, two other possibilities

```
\def\ShowOffMathMacrosInDisplayStyle{%
  \ValueTemplate={$\displaystyle{##}$}}
\def\ShowOffOrdinaryMacros{%
  \ValueTemplate={##}}
```

that weren't mentioned in the introduction. Specifying `\ShowOffMathMacrosInDisplayStyle` causes math macros to be exhibited in display style so that, for example,

```
\ListMacros
  \def\SumA{\Sum i 1 n}
  \def\Sum#1#2#3{\sum_{#1=#2}^{#3}}
\DontListMacros
```

will produce

`\SumA`	$\displaystyle\sum_{i=1}^{n}$	`\Sum i 1 n`
`\Sum#1#2#3`	$\displaystyle\sum_{\#1=\#2}^{\#3}$	`\sum _{#1=#2}^{#3}`

instead of the default

`\SumA`	$\sum_{i=1}^{n}$	`\Sum i 1 n`
`\Sum#1#2#3`	$\sum_{\#1=\#2}^{\#3}$	`\sum _{#1=#2}^{#3}`

Non math-mode macros (e.g., abbreviations for long words) can be exhibited by specifying `\ShowOffOrdinaryMacros`.

◇ Michael J. Wichura
Department of Statistics
Computation Center
University of Chicago
5734 University Avenue
Chicago, IL 60637
wichura@galton.uchicago.edu