Paradigms: Two-part macros

Kees van der Laan

Hunzeweg 57,
9893 PB Garnwerd, The Netherlands
cgl@rc.service.rug.nl

1 BLUe's Design III

Hi folks. When attending Amy's class in 1990, I was much surprised about the two-part macro. In Algol, FORTRAN, PASCAL and ADA, I had not heard of the concept, let alone I was familiar with it.

In blue.tex they are at the heart of the syntax for the markup language. To speak with Jackowski 'I use them all the time.'

2 One-part vs. two-parts

One-part macros are explained in chapter 20 of *The TEXbook*. They are used as a shortcut for the replacement text parameterized by at most nine arguments.

A two-part macro is different. The first part sets up the 'environment' followed by script elements and ended by the second part, to finish up the environment. LaTeX emphasizes the environment concept in for example

```
\begin{abstract}...\end{abstract}
\begin{center}...\end{center}
\begin{itemize}...\end{itemize}
\begin{picture}...\end{picture}
\begin{quote}...\end{quote}
\begin{tabular}...\end{tabular}
\begin{thebibliography}...
          \end{thebibliography}
\begin{verbatim}...\end{verbatim}
%etc.
```

3 Why?

The need for bothering about two-part macros is that the enclosed script elements are processed on the fly, meaning with the right catcodes.

To digress a little on the above the following hypothetical example. Suppose we have

```
{\catcode`*=13
 \gdef\begindemo{\bgroup
   \catcode`*=13 \def*{MUL}}
\gdef\demo#1{\catcode`*=13 \def*{MUL}#1}
}\let\enddemo\egroup
```

then the result of

```
\begindemo*\enddemo
%and
\demo*
```

is different. The first yields MUL and the latter *.

Explanation

In the two-part case the * is seen after the catcode has changed. while in the latter the * is seen, and the catcode *fixed*, before it is made active.

However, in chapter 20 of *The TEXbook* there is no treatment of two-part macros, nor is there an entry for it in the index, alas. Exercise 5.7 deals with named blocks and checking of them. The latter is used in LaTeX to make sure that the right environment closing tag is used in the markup. In Appendix E, where example formats (o.a. manmac) are explained, two-part macros are abundant, for example

```
\beginchapter...\endchapter
\beginlines...\endlines
\begindisplay...\enddisplay
\begintt...\endtt
\begimathdemo...\endmathdemo
\beginchart...\endchart
\beginsyntax...\endsyntax
\begindoublecolumns...\enddoublecolumns
\exercise...\answer...\par
```

Furthermore, of late two questions were posed on TeX-nl, which exposed the unfamiliarity with two-part macros. All this was enough for me to spend a paradigm column on two-part macros.[1]

Example (`\beginlines...\endlines`)

The functionality is that the script in between is processed line-by-line and preceded and followed by an `\hrule`.

```
\def\beginlines{\par\begingroup\nobreak
   \medskip\parindent0pt\hrule\kern1pt
   \nobreak\obeylines\everypar{\strut}}
\def\endlines{\kern1pt\hrule\endgroup
   \medbreak\noindent}
```

In the TEXbook script this is combined with in-line verbatim.[2]

Explanation

The replacement text of `\beginlines` is processed, followed by the formatting on-the-fly of the inserted material (after `\beginlines`) up to `\endlines`. The replacement text of the latter finishes it up.

[1] Note that `\beginchapter`'s title is not processed on the fly. In the 'Paradigm: Headache?' I have shown how the title and the contents of the chapter can be processed on the fly.

[2] To set text verbatim. By the way, this is another approach to 'verbatims with an escape character.'

Unwanted breaks are avoided. The `\hrule` is set in the first part and in the second part next to opening and closing of the group. The in between script is processed with `\obeylines` on.

Example *(*`\begindisplay`. . . `\enddisplay`*)*

The functionality is that the script in between is processed as a non-centered display, indented by `\displayindent`, next to the value of `\parindent` from the template. Pruned from non-essential issues for the two-part macro idea, the macros read as follows.

```
\def\begindisplay{$$\the\thisdisplay\halign
   \bgroup\indent##\hfil&&\qquad##\hfil\cr}
\def\enddisplay{\crcr\egroup$$}
```

Explanation

The replacement text of `\begindisplay` is processed, followed by the formatting on-the-fly of the inserted material (after `\begindisplay`) up to `\enddisplay`. The replacement text of the latter finishes it up.

$$ followed by `\halign` is something special. It starts the so-called alignment display, meaning that each hbox of the `\halign` is added to the main vertical list indented at the left by `\displayindent`. It is *not a math display*. `\the\thisdisplay` allows to insert assignments.

By the way, note that the user is not bothered by the details of the template of the `\halign`; it is already there.[3]

And what about a one-part on top?

This is not possible via my method as explained in 'Paradigms: Headache?', because each table entry must have balanced braces. Suppose we have

```
\def\display#{\begindisplay\bgroup
   \aftergroup\enddisplay
   \let\dummy=}
```

then the `\bgroup` after `\begindisplay` is 'unbalanced' in the first column, except when it is about one entry only.

```
\display{a}  %works
\display{a&b}%doesn't work
```

I let it go

because I could not provide a nice solution. What I tried is out of balance with just using the two-part macros. The best I could get at, when we allow in-line verbatim, needs the following input.

```
\thisdisplay{\catcode`\!=0 \catcode`\\=12 }
\display{\a&b!cr e&f}
```

Conclusion

When tables are involved my method of building one-part macros on top of two-part macros is not suited.[4]

For the manmac

version of the two-part macro see *The TEXbook* 421. Note that there the `\catcode`\^^M` annihilates the effect of `\obeylines`. The `\obeylines` was introduced only to allow for an optional argument. Because of my `\thisdisplay` toks variable, the `\obeylines` and its annihilator are no longer needed. Knuth's coding has been simplified, at the expense of introducing a token variable `\thisdisplay`.[5]

4 From the TeX-nl list

Andrea de Leeuw van Weenen and Ton Biegstraaten posed the following problems.

- let characters print other characters
- let _ in math denote an underscore and not a subscript.

Although it turned out that my suggestions are not the 100% required ones, I'll expose them here nonetheless, because they illustrate the use of two-part macros.

Andrea's problem

Let us suppose that the problem is to let B typeset 1, on demand. Then a solution reads.

```
\def\beginIT{\bgroup\catcode`\B=13 \ITstart}
{\catcode`\B=13
 \gdef\ITstart{\def B{\char'61}}}
\def\endIT{\egroup}
%with use
ABC\quad
\beginIT ABC\endIT\quad
ABC
```

The result reads ABC A1C ABC.

The problem which remained is that Andrea needs simultaneously macros with those letters like B in their name. She added the problem to her list of 'Impossible with TEX problems.[6]'

Ton's problem

The restriction, which made that my solution was not appropriate, is that it should be possible to use the solution as argument of one-part macros, and that to unlimited depth.[7] In my approach all involved one-part macros had to be rewritten into two-part ones. However, if people would start to think in two-part macros (nearly) all would have been fine.

```
\def\beginusn{\hbox\bgroup\catcode`\_=13
   \startusn}
```

[3] In my `\btable` macro, I allowed the possibility for a user to supply his own template, because I stored the template in a token variable.

[4] If one prefers a simple, but *restricted* one-part macro provide `\def \display`#1{ `\begindisplay` #1 `\enddisplay`}.

[5] Optional arguments — well, more generally 'Parameterization' — will be subject of the next paradigm column.

[6] I'm curious to see that list in MAPS some day.

[7] Courtesy Piet van Oostrum.

```
{\catcode‘\_=13\gdef\startusn{\def_{\_}}}
\def\endusn{\egroup}
%with use
$a_b\quad \beginusn a_b\endusn\quad a_b$
```

and result

a_b a_b a_b.

On top of the above two-part macros we can add one-part macros *wih the same functionality*, as explained in the ‘Paradigm: Headache?’

The one-part macros read `\def\IT{\beginIT\bgroup`

```
   \aftergroup\endIT
   \let\dummy=}
%
\def\usn{\beginusn\bgroup
   \aftergroup\endusn
   \let\dummy=}
```

As expected `ABC\quad\IT{ABC}\quad ABC`
yields ABC A1C ABC, and
`$a_b\quad\usn{a_b}\quad a_b$`
yields a_b a_b a_b.

Note that I omitted here the # as last element of the parameter list, neglecting some built-in security checks.[8]

5 `\eqalign` as two-part macro

As an example of how to cast a one-part macro into two parts, and a one-part macro[9] on top, let us rewrite `\eqalign`, *The TEXbook* 362. The extra functionality of this approach is that the two-part variant can be used in those cases where the argument needs to be processed on the fly.

```
\def\begineqalign{\,\vcenter\bgroup
   \the\thiseqalign\openup1\jot\m@th
   \starteqalign}
\def\starteqalign{\ialign\bgroup
   \strut\hfil$\displaystyle{##}$&&
   $\displaystyle{{}##}$\hfil\crcr}
\def\endeqalign{\crcr\egroup\egroup}
%with the one-part
\def\eqalign#1{\begineqalign
   #1\endeqalign}
```

I don’t have a concrete example for the need for modifying `\eqalign` towards processing on the fly. However, it illustrates how to rewrite a one-part macro into two-parts as basis.

Looking back

I like the consistent markup via

`\begin`⟨*tag*⟩ ⟨*copy proper*⟩ or `\`⟨*tag*⟩`{`⟨*copy proper*⟩`}`
`\end`⟨*tag*⟩

The right-hand variant is suited for the markup of headings, for example. It has been adopted in blue.tex, as basic syntax, for the markup language.

6 Multiple use of copy

Sometimes we need to process the copy — or should we talk about data then? — more than once. An example is the data for a crossword, where I used the data for typesetting the puzzle — the data reflect the structure — and the solution. See ‘Typesetting crosswords via TEX, revisited,’ MAPS 92.2.

The basic idea is to store the data with the right catcodes.[10]

```
\def\bdata{\begingroup
   \obeylines\obeyspaces\store}
\def\store#1\edata{\endgroup
   \def\storeddata{#1}}
```

Explanation

The data, in natural markup line-by-line, can be supplied between `\bdata` and `\edata`. The `\edata` is a parameter separator and not the invocation of the closing part of a two-part macro, although it looks the same. What happens is that `\bdata` sets up the environment, especially provides the right catcodes. `\store` ends the environment (scope) and stores the data, with the wanted catcodes, as replacement text of `\storeddata`. In order to appreciate the subtleness of the above coding the following digressions.

6.1 Two-part macros and storing on the fly

This is inhibited by the following[11]

- the *opening and closing brace* of the replacement text of a `\def` must be explicit
- the right-hand side of a token list assignment must be explicit.

The following innocent coding is therefore incorrect.[12]

```
\def\bdata{\begingroup
   \obeylines\obeyspaces
   \gdef\storeddata\bgroup}
\def\edata{\egroup\endgroup}
```

Possible alternatives to my coding above are

```
\def\data{\obeylines\obeyspaces
  \gdef\storeddata}
%with use
\begingroup
\data{ab c
```

[8] In the case of the # end separator the text after the macro invocation must syntactically begin with an opening brace. When the # separator is omitted, anything can follow ⟨tag⟩.

[9] Not more limited than the one available.

[10] Perhaps the most trivial approach is to insert the data each time we need it. I consider that inelegant and also error-prone. The given macro is a beautiful example, if I may say so, of what Victor Eijkhout and David Salomon call two-step macros (see later), while at the user level the macro can be used as if it is a two-part macro, with the nice opening and closing tags.

[11] Courtesy Victor Eijkhout in ‘TEX by Topic,’ section 10.3 group delimiters. Awareness of these restrictions is indispensable for writing two-part macros. I omitted the use of `\setbox`, because once set in a box one can’t do much with the data anymore.

[12] The use of explicit braces is incorrect as well.

```
      e fg}
\endgroup
%
%and via the use of a toks variable
%
\newtoks\storeddata
\def\data{\obeylines\obeyspaces
  \global\storeddata}
%with use
\begingroup
\data{ab c
      e fg}
\endgroup
```

Nice aspects of the above approaches are

- at the outer level I abstracted from storing in a def or a token variable, and
- the symmetry.

Definitely not nice aspects are

- it looks as if the data are stored in `\data`, and
- the `\begingroup` and `\endgroup` at the user level.

6.2 A one-part on top?

My scheme does not work for this case. Some puzzling yielded as one-part `\data` on top of `\bdata`, with `\edata` eliminated.[13]

```
\def\data{\begingroup
    \def\store##1{\endgroup
       \gdef\storeddata{##1}\endgroup}
    \bdata}
%With use
\data{ab c
      d ef}
```

Explanation

`\data` starts a group and (re)defines `\store`. The invocation of `\bdata` set the catcodes — via `\obeylines` and `\obeyspaces` — and invokes `\store`. The argument to the latter macro is stored in `\storeddata` with the right catcodes. `\store` also ends the groups.[14]

6.3 Chapterhead

For blue.tex I designed `\report`. A report takes chapter titles. The problem is: How to write macros consistent with the philosophy of starting from two-part macros and building a one-part on top, *with* the chapter title also stored for use in the running headline, for example.

In an abstract sense this is equivalent to the `\bdata` `\edata`, `\data` suite. It is even simpler, because I just have to store the name and allow the following use.

```
\beginchapterhead
<name>            or    \chapterhead{<name>}
\endchapterhead
```

The required result must be such that the chapter name will be typeset appropriately within context, as prescribed by the token variables `\prechapterhead` and `\postchapterhead`, and that the name will be stored in the token variable `\chaptername`.

The coding of the two-part macros read.

```
\def\beginchapterhead{\the\prechapterhead
   \storechaptername}
\def\storechaptername#1\endchapterhead{%
   \chaptername={#1}\endchapterhead}
\def\endchapterhead{{\chpfont
   \the\chaptername}\the\postchapterhead}
```

The one-part macro on top reads.

```
\def\chapterhead{\bgroup
   \def\storechaptername##1{\egroup
     \global\chaptername={##1}%
     \endchapterhead}
   \beginchapterhead}
```

The head-suite of macros also need processing and storing if not for writing to a ToC file. The use of the token variable `\prechapterhead` provided the hook to change the catcode of the circumflex — which in blue.tex is default active because of preparing Index Reminders — into 7 and allow processing math as part of the title.

What have we gained?

We can use now the title with different fonts, as title and in the running head. Moreover, we can use the `\beginchapterhead`, `\endchapterhead` pair to enclose the title, or let it look as an assignation to `\chapterhead`. Looking back there emerged a paradigm for the use

```
\begin<tag>
<copy>            or    \<tag>{<copy>}
\end<tag>
```

with `<copy>` also stored in the token variable `\tagname`. Useful!

6.4 And what about multiple use with different catcodes?

Like Knuth we are at loss, unless we make use of a file. It occurs in manmac's math demos, for example

Input	*Output*
`$x^2$`	x^2

needs markup with repetition of the data[15]

```
\beginmathdemo
   \it Input&\it Output\cr
   \noalign{\vskip2pt}
   |$x^2$|%<---
   &x^2   %<---
\endmathdemo
```

Subtle, very subtle. One thing is crystal clear, however. Because of the above varieties (and pitfalls?), a discipline of TEX coding is needed.

[13] Note that in-line verbatim as part of the data goes wrong, in the sense of unexpected results.

[14] The group opening in `\bdata` is not needed here, but within the context of the two-part macro next to the one part, it is needed.

[15] Borrowed from *The TEXbook* script. In blue.tex I added `\crcr` to `\endmathdemo`, for consistency with `\halign` use.

7 Epilog

Eijkhout in 'TEX by Topic' and Salomon in 'Insights and Hindsights' treat *two-step* macros, not *two-part* macros.[16] One macro will set up conditions and a second will do the work. The difference with two-part macros is that the 'workmacro' also terminates the conditions, while in two-part macros the second part has only the functionality to terminate. Probably other macros are involved to do the work. A beautiful example from manmac is the non-centered display macro with tags

```
\begindisplay %to set up conditions
\startdisplay %to do the work
\enddisplay   %to finish up
```

As known, I prefer — like Knuth — the separation of concerns principle, and like opening and closing tags.

To my knowledge it is not possible to build gracefully, and with the *same functionality*, a one-part *table* macro on top of its constituent two-parts, in full generality.

Have fun, and all the best.

[16] Apparently they did not inspect manmac in detail. In Eijkhout's book look at section 11.9.4, the macro `\PickToEol`. In Salomon's courseware look at section 5.19, the macro `\elp`.