"Hey — it works!"

Jeremy Gibbons

Welcome again to *"Hey — it works!"*, a column devoted to useful or surprising (LA)TeX and META techniques. I am writing this time from TUG2000, ably organized by Sebastian Rahtz and Kim Bruce in my home town of Oxford (from which you may conclude that my timekeeping has not made our noble editor's life any easier — sorry, Barbara!).

In this issue we have three items: one describing a macro of unknown provenance for yielding a non-punctuating comma for a decimal point, a slightly related one inspired by André Van Ryckeghem from Belgium on currency conversion, and one by Pedro Palao Gostanza from Madrid on counting the number of parameter uses in the expansion text of a macro.

◇ Jeremy Gibbons
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
jeremy.gibbons@comlab.ox.ac.uk
http://www.comlab.ox.ac.uk/oucl/people/jeremy.gibbons.html

1 Decimal comma

In some languages, a comma is used instead of a full stop to separate the whole and fractional parts of a decimal number: 3,1415. Unless one does something special about it, TeX prints instead $3, 1415$, treating the comma as punctuation rather than an ordinary symbol and so putting some space after it.

Someone recently posted the following style file to `comp.text.tex`[1]:

```
\mathchardef\ocomma="013B
\mathchardef\pcomma="613B
\mathcode`\,="8000
{\catcode`\,=\active
 \gdef,{\obeyspaces
   \futurelet\next\smartcomma}}
\def\smartcomma{\if\space\next
  \pcomma\else\ocomma\fi}
```

This style file defines two different macros to generate a comma, one as an ordinary symbol and one as a punctuation. The comma is then made active in maths mode, and defined to look ahead to the next token; if this next token is a space, a punctuation comma is used, and otherwise the ordinary comma is used. Thus, `3,1415` yields the decimal number 3,1415, whereas `(3, 4)` yields the pair $(3, 4)$.

It strikes me that it would be better to determine whether the next token is a digit, and only to use the ordinary comma when it is; then it does not matter whether a space is used in, say, (x, y). But with a little experimentation I could not make this work; can anyone help?

2 Two decimal digits

On `comp.text.tex` recently, André Van Ryckeghem asked how to perform currency conversion. He already had the translation working on integer values, with a macro like the following:

```
\def\convert#1#2{{% evaluate #1/#2
  \count0=#1\relax % numerator
  \count1=#2\relax % denominator
  \count2=\count1 \divide\count2 by 2\relax
  \advance\count0 by \count2\relax
  \divide\count0 by \count1\relax
  %
  \the\count0\relax
}}
```

The main point here is how to divide one value m by another n and round the result to the *nearest* integer, instead of the usual rounding down to the *next smaller* integer; this is done by computing instead $(m+n/2)/n$ and rounding down as usual. For example, at the time of writing, the exchange rate is ¥162 to £1, so with

```
\convert{34567}{162}
```

we learn that ¥34567 is approximately £213.

André's problem was to incorporate decimal output. For example, as it stands we learn only that ¥345 is approximately £2, which is not very accurate. On the other hand, real number arithmetic is inappropriate, because we do not want umpteen digits after the decimal point (£2.12962). Instead, we define

```
\def\convert#1#2{{% evaluate #1/#2, to 2dp
  \count0=#1\relax
  \multiply\count0 by 100\relax % 100*numerator
  \count1=#2\relax % denominator
  \count2=\count1\relax
  \divide\count2 by 2\relax % half denominator
  \advance\count0 by \count2\relax
  \divide\count0 by \count1\relax % rounded divn
  %
  \count3=\count0\relax
  \divide\count3 by 100\relax % whole part
  \count4=\count3\relax
  \multiply\count4 by -100\relax
  \advance\count4 by \count0\relax % frac part
```

[1] Unfortunately, the identity of the original author has been lost. If it was you, let me know and I'll give an update in my next column. The original was called `komma.sty`, dated 1998/06/28, and had some `@` symbols in the macro names.

```
  %
  \the\count3.%
  \ifnum \count4<10\relax 0\fi
  \the\count4\relax
}}
```

The difference is that we work throughout with two more significant figures (that is, we multiply the numerator by 100). To print the result, we finally divide by 100 again to print the whole part, then print the remainder (possibly with a leading zero, if it is just a single digit) to get the fractional part. With this definition, we find that ¥345 is more accurately £2.13.

Note that all arithmetic is performed with TEX's 32-bit integers, so is limited to about nine digits. There are strong arguments for using arbitrary precision integers for financial computations. André observes that a much more elaborate and robust solution than this is provided by Melchior Franz' `euro` package, which performs and typesets conversions between arbitrary currencies, using Michael Mehlich's `fp` package for exact arithmetic with 36 significant digits (both packages being available on CTAN).

3 Number of parameter tokens

While doing some meta-macros I was in need to count how many parameter tokens appear in a definition parameter text. A complete solution would need the capability of TEX to match against a `{`.

It is easy to work with parameter texts if they are stored in *saturated* macros: macros with nine undelimited parameter tokens. The three following saturated macros containing parameter text will be used as a running example.

```
\def\pp#1#2#3#4#5#6#7#8#9{%
  #1trivial#2parameter#3}
\def\qq#1#2#3#4#5#6#7#8#9{%
  #1\undefined#2parameter#3}
\def\kk#1#2#3#4#5#6#7#8#9{%
  #1problem#2\gobbledisttag#3}
```

In the rest of this note, *parameter text* usually means parameter text stored in a saturated macro. The goal is to define a macro `\nop` returning in a counter the number of parameter tokens in a parameter text; the counter and the parameter text are, in this order, the only arguments of `\nop`.

The main idea is simple: substitute each parameter token for a counting code like

```
\advance\counta by 1
```

The following macro allows us to put the same thing in each parameter token

```
\def\applyall#1#2{#1%
  {#2}{#2}{#2}{#2}{#2}{#2}{#2}{#2}{#2}}
```

The difficult part is to throw away all the material between the parameter tokens; we will call it *disturbing* material. My first solution was really simple and worked almost always.

```
\def\nop#1#2{#1=0
  \expandafter\expandafter\expandafter
    \gobbledist\applyall#2%
    {\gobbledisttag\advance#1by1
     \gobbledist}%
    \gobbledisttag}
\def\gobbledist#1\gobbledisttag{}
```

But this fails if `\gobbledisttag` appears in the parameter text; that is, `\pp` and `\qq` parameter texts are counted without any problem, but `\kk` raises an error because of a dangling `\gobbledisttag`. Although this is not a practical restriction, it is an aesthetic one. To cope with it we need to mark the beginning of a parameter token with something that cannot be used in parameter text: outer macros, `{`, `}` or `#`. But if the mark cannot be used in a parameter text, neither can it appear in the definition of `\gobbledist` (1). So, a trick different from a *gobble* macro is needed to throw away the material between parameter tokens (2).

Expanded definitions (`\edef`) are a good place to look next. In cooperation with brace hacks from the TEXbook, allow us to put disturbing material inside braces.

```
\def\nopB#1#2{#1=0
  \expandafter\expandafter\expandafter
    \voidefdist\applyall#2%
    {\voidefdistend\advance#1by1
     \voidefdist}%
    \voidefdistend}
\def\voidefdist{\edef\aux{\iffalse}\fi}
\def\voidefdistend{\iffalse{\else}\fi}
```

Disturbing material ends in the definition of the `\aux` macro. But this solution is even worse because TEX does not allow undefined token in an expanded definition; both `\qq` and `\kk` give errors because neither `\undefined` nor `\gobbledisttag` are defined. If the parameter text has conditionals, errors can happen far beyond `\nop`.

Neither normal definitions, nor token lists help in this problem, because they need balanced braces; they surely solve it completely if combined with the `\scantoken` extension of ϵ-TEX. Boxes definitions are worse than expanded definitions.

It took me some time to realize that, in TEXory, (2) cannot be derived from (1) because TEX allows to match against tokens that cannot appear in a parameter text: if a parameter text ends with `#`, TEX will match against an open brace (*The TEXbook*,

p. 204). This observation turned me back to search for a *gobble*-like solution.

```
\def\nop#1#2{#1=0
  \expandafter\expandafter\expandafter
    \gobbledist\applyall#2%
    {{}\advance#1by1 \gobbledist}%
    {}}
\def\gobbledist#1#{\gobble}
\def\gobble#1{}
```

This solution is a bit strange because usual TEX practice dictates that, in order to catch something, it should be surrounded with `{`$\cdots$`}` but, in some sense, we are surrounding the disturbing material with `}`$\cdots$`{`.

Although the last definition of `\nop` is aesthetically more pleasant than the first one, it has its same drawback: `\nop` cannot count how many parameter tokens has `\gobbledist`! I am sure that the reader will find a nice fix.

◇ Palao Gostanza, Pedro
Universidad Complutense de Madrid, Spain
ecceso@sip.ucm.es