Macros

The bag of tricks

Victor Eijkhout

Hello all. The other day I was asked to assist in solving the following problem: format a file of address labels that is given as plain text, like

```
My Name
123 My Street
My Town
```

with the items of a label on separate lines and the labels separated by empty lines. Without inserting any TEX commands, of course.

This problem turned out to be a tricky one, and I'll make this into a sort of a tutorial on end-of-line handling in TEX[1].

Normally, TEX converts the end of the line into a space, but here we want to keep the lines the way they are. For this, we start mucking about with the 'secret character' that TEX puts at the end of each line. Every time TEX reads a line from the input file, it appends the character with number `\endlinechar`[2]. Usually, this is character 13, and you can write `'\^^M` if you don't want to remember that number[3].

The crucial point is that this character has category code 5, for end-of-line. TEX converts such characters into spaces, or into `\par` if it finds them on an otherwise empty line[4].

And now we are in trouble: on the one hand you would want to write a macro along the lines of

```
\def\OneLabel#1\par{...}
```

but that `\par` token needs an `\endlinechar` of category code 5, and that would obliterate the line ends, which we wanted to keep.

Looking at it from the other side, if we change the category code of the line end to anything but 5 in order to keep it recognizable, we don't get a `\par` token after the label text anymore.

There doesn't seem to be another possibility than changing the category code of the line end, and processing the labels line by line.

Let us start programming bottom-up with some preliminaries: we will need macros to process the labels. I assume that you define macros `\AddToLabel` and `\LabelFinished`, for instance like this:

```
\newbox\LabelBox
\def\LabelFinished
  {\box\LabelBox}
\def\AddToLabel#1%
  {\setbox\LabelBox
     \vbox{\unvbox\LabelBox
           \hbox to 5cm{#1\hfil\strut}}
  }
```

Now we continue top-down by specifying how the formatting is going to look to the user. Here is how it could be done in plain TEX.

```
\def\endplainlabels{\Bye}
\plainlabels{Here come the labels:\par}

My Name
My Street 1
My Town

Your Name
Your Street 2
Your Town

\Bye
```

The command `\endplainlabels` specifies how TEX recognises that the labels are finished. Whatever is on that line is also executed. The `\plainlabels` command[5] starts the formatting of the labels, and its argument (which can be empty) specifies whatever should be done prior to typesetting the labels. The blank lines before the first and after the last label are optional.

One remark: the `\bye` and `\end` macros are outer macros, so you cannot write

```
\def\endplainlabels{\bye}
```

Instead you have to resort to the following trick:

```
\edef\endplainlabels{\noexpand\bye}
```

Here is the full implementation of the line processing macros. I am assuming that you will put them into a separate file

[1] And I'll delegate all the 'unlesses' to the footnotes. If you want to read about this topic in more detail, read chapter 2 of my book 'TEX by Topic'.

[2] Unless this number falls outside the range of 0–255.

[3] So why is `'\^^M` easier to remember? Well, in Ascii, <Control>-M is the Carriage Return. Does that help?

[4] It doesn't hurt if your input file has some spaces on the 'empty' line, because spaces at the end of a line are discarded. There is also something about spaces at the beginning of a line, but that's a different story.

[5] I've tried to make this into a LATEX environment, but ran into all sorts of problems. Sorry. Simply use the same syntax in LATEX as in plain TEX.

```
\endlinechar=-1
\def\empty{}
\newif\iflabelpending
\catcode`\^^M\active
\long\def^^M#1^^M{\def\test{#1}
  \ifx\test\endplainlabels
    \catcode`\^^M=5\relax
    \iflabelpending\LabelFinished\fi
    \expandafter\endplainlabels
  \else \ifx\test\empty \LabelFinished
          \labelpendingfalse
        \else \AddToLabel{#1}
          \labelpendingtrue
        \fi
        \expandafter ^^M
  \fi
  }
\long\def\plainlabels#1
  {\toks0{#1}\labelpendingfalse
   \edef\next{\everypar
    {\the\everypar
     \everypar{\the\everypar}
     \the\toks0\relax
     \catcode`\noexpand\^^M\active
     \noexpand^^M}}
   \next\par}
\endlinechar=`\^^M \catcode`\^^M=5 \relax
\endinput
```

There are lots of tricky points to these macros. Here are a few

- All that redefining of `\everypar` is for the benefit of packages such as LaTeX which themselves redefine `\everypar`. The macros given here make sure that the custom `\everypar` first executes whatever was in the old one, and after executing its own commands, restores the old value.
- The conditional `\iflabelpending` handles the case where there is no blank line after the last label. Without it, that label would not be printed.
- The `\par` at the end of `\plainlabels` puts TeX into vertical mode, so that the first label will trigger `\everypar`.
- The `\toks0` register makes sure that the initial commands get executed after TeX has come out of vertical mode: this is mostly for the case of LaTeX lists; see the examples below. They do not like it if an item occurs in vertical mode[6].

For a slightly more complicated example, let us turn the labels into items in a LaTeX list. Define

```
\def\LabelFinished
  {\item[]\box\LabelBox}
```

and process the labels with

```
\def\endplainlabels{\end{trivlist}}
\plainlabels{\begin{trivlist}}
...
\end{trivlist}
```

Would you like to have several labels on one line? Use the following macros:

```
\newbox\AllLabels
\def\LabelFinished
  {\setbox\AllLabels\hbox
     {\unhbox\AllLabels\hfil
      \box\LabelBox}}
\def\ejectlabels{\linepenalty100
  \noindent
  \unhbox\AllLabels}
```

This appends all labels to a long `\hbox`, which you'll have to eject at the end — and it is then treated as a paragraph — with

```
\def\endplainlabels
  {\ejectlabels\end{something}}
\plainlabels{Something something}
\ejectlabels\end{something}
```

If you catch my drift. And you may want to make sure that all labels have the same width:

```
\def\AddToLabel#1%
  {\setbox\LabelBox
    \vbox{\unvbox\LabelBox
          \hbox to 3cm{#1\hfil\strut}}
  }
```

Finally, a comment for the true hackers among you: suppose you don't want to assume that the `\endlinechar` is 13. First of all you'll have to add a few lines:

```
\count0=\endlinechar \endlinechar=-1
...
\endlinechar=\count0
```

to save and restore the proper `\endlinechar` while you define the macros. More importantly, you don't know what active character to define for processing the lines! Here's a way out:

```
\begingroup
\uccode`\^^N=\count0 \catcode`\^^N\active
\uppercase{\gdef^^N#1^^N}{....}
\endgroup
```

This uppercases an active character 14, and you've set it up so that this is the (saved) end-of-line character.

Happy hacking to you hackers, and to the rest, don't be afraid to ask, we hackers are only too happy

[6] This is also the reason that I could not use grouping: LaTeX's tests have to be set globally.

to show off. And reader contributions for this column are still welcome!

◇ Victor Eijkhout
Department of Computer Science
University of Tennessee at
Knoxville
Knoxville TN 37996-1301
Internet: eijkhout@cs.utk.edu

Random Bit Generator in TEX

Hans van der Meer

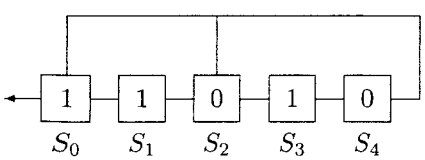
1 Introduction

When I started using TEX for my collection of exam questions, the need of a random bit generator arose. With such a generator it is easy to randomly permute items of multiple choice questions, choose between different variants, etc.

Since part of my interests are in the field of cryptography it was most natural to look for a convenient source of a random bitstream in that field. Such a source is provided by shiftregisters, the simplest form of which is the linear variety. Although not strong enough for direct use in cryptographic applications, their random properties are nevertheless excellent. Furthermore they are easily implemented, a real asset because of TEX's limited abilities in arithmetic. The prime reference for shiftregisters is the famous book by Golomb[1].

2 Linear Shiftregisters

Before describing how such a shiftregister can be implemented in TEX, it is necessary to have a modest look at their construction. The figure shows a small linear shiftregister. It consists of five so-called *stages* $S_0 \ldots S_4$ and is therefore called a five-stage register. Each stage is a memory unit capable of holding one bit. The values of all the stages together make up the *state* of the register; in the figure the current state $S = (11010)$.



The register is operated in the following way. At each step the bits in the stages are shifted to the stage at their left. The bit in stage S_0 is thereby produced as the output bit. Of course the vacancy left in the rightmost stage must be filled up. Therefore all stages which in the figure have an exit at the top of the stage box, also spawn their bit through this exit just before the bit migrates to the left. These exits are called *taps*. The bits spawned are combined by the exclusive-or operator and the resultant bit fills the rightmost stage. E.g., with taps at $S_i, S_j, S_k, \ldots$ the mod 2 sum $S_i \oplus S_j \oplus S_k \oplus \ldots$ is formed. Thus the register produces an output bit and a new state at each operation step. In the example the output bit will be a 1 and the next state $S = (10101)$.

It is easily understood that eventually the bitstream must repeat itself. Because an n-stage register holds an n-bit quantity it can exist in 2^n different states only. Since new states are produced by a strictly deterministic process, a periodic pattern of successive states must result. Thus the output stream will be periodic. Of course it is desirable that the length of the cycle be as long as possible.

These registers can also be described with a polynomial in a bit variable $x \epsilon \{0, 1\}$, called the *characteristic polynomial*. The example register has characteristic polynomial

$$f(x) = 1 + x^2 + x^5$$

It turns out that the length of the cycle produced by a register characterized by a given polynomial is connected to certain properties of this polynomial. Particularly useful are the so-called primitive polynomials.[1] One is able to show that primitive polynomials lead to the longest possible period for a linear shiftregister of a given size. In fact two cycles are produced: (1) a cycle of period 1 consisting of a stream of zeroes, (2) a fine random stream of zeroes and ones of length $2^n - 1$. The first cycle, the zero cycle, is not entirely useless as it offers a natural way for shutting off the random stream.[2]

After having explained how a shiftregister works, it is easy to see why I chose the register based on

$$f(x) = 1 + x^{21} + x^{22}$$

for the implementation of a random bit generator in TEX. It is a primitive polynomial and therefore has a longest period of 4,194,303 bits — more than enough for all but the most exotic applications. And another important fact is that it has only two taps,

[1] Roughly the equivalent of a prime number among polynomials plus an additional condition.

[2] I am using this stream when typesetting the full collection of exam questions. The absence of random shuffling makes it easier to connect the printed output with the TEX input.