Generating \n asterisks

George Russell

At the start of Appendix D of *The TEXbook*, Donald Knuth considers the "toy problem" of defining a macro \asts which contains just \n asterisks (where \n denotes one of TEX's count registers). For simplicity I intend to assume that \n is non-negative from now on, though ideally we should check that \n is a count register with non-negative contents before executing the macros.

The first solution by Knuth uses the following operations:

(a) we can make \asts null with \xdef\asts{};
(b) we can add one asterisk to it with \xdef\asts{\asts*}.

Therefore we just make \asts null and then add one asterisk to it \n times. Unfortunately TEX needs $O(t)$ time to execute step (b) when \asts contains t tokens, so the whole method takes $O((\backslash n)^2)$ time in total.

However it would be much nicer to have a solution which took time proportional to \n rather than its square. Knuth gives one. This works by building up a definition for \asts on the TEX save stack using \aftergroup. But the problem with this solution is that as we rarely require a large save stack, most implementations only have a small one, so Knuth's solution probably will not work for large \n (on the TEX I usually use, it fails for \n bigger than 170 or so*).

Therefore I propose two refinements (in the order they occurred to me) which are both linear and allow \n to get quite large.

The first refinement can be thought of as follows. If we look again at the list of operations which were used for Knuth's first solution, we see that there is a third useful operation which can be used to increase the size of \asts; namely:

(c) we can *double* \asts using \xdef\asts{\asts\asts}.

Thus if we want a 1000-asterisk macro, we can generate it by generating in turn 0-, 1-, 2-, 3-, 6-, 7-, 14-, 15-, 30-, 31-, 62-, 124-, 125-, 250-, 500- and 1000-asterisk macros, obtaining each from the previous one by adding an asterisk with (b) or doubling with (c) (this method is analogous to the algorithm for taking powers by repeated squaring). Here therefore is my first solution to Knuth's problem, which (as the reader can satisfy himself) is linear in \n.

```
\def\makeasts#1{{% Function is to make
%            \asts contain \n asterisks.
  \count0=#1 % Put argument into a
% register so we can do arithmetic on it.
% (it is OK to use \count0 and \count2
% as scratch registers as no output is
% generated in the macro so they will
% not be referenced by \output.)
  \ifnum\count0=0 %
    \xdef\asts{}% operation (a)
  \else
    \count2=\count0\divide\count2 by 2
%          Set \count2 to half \count0.
    \makeasts{\count2}%
    \xdef\asts{\asts\asts}% operation (c)
    \ifodd\count0
      \xdef\asts{\asts*}\fi% operation (b)
  \fi}}
\makeasts{\n}
```

This solution is reasonably fast, and works with \n as large as 39800 on my local implementation (because it is limited by the size of TEX's main memory rather than the size of its save stack). Only a maniac would want more asterisks than that! Of course it will fall over for smaller values of \n if we have other stuff occupying the main memory (the figures given here were obtained on a version of TEX with only the Plain TEX macros loaded). So I was reasonably satisfied, until Chris Thompson, the local TEX wizard, asked me the following question: is there a set of TEX commands which expands to \n asterisks without using any primitive commands? (The tokens TEX understands can be divided into those which reach its stomach, like the "typeset character" commands and \def (described in chapters 24–26 of *The TEXbook*) and those which are expanded and removed in its mouth, like macros, \if and \the (described in chapter 20). It looked as if the answer to Chris Thompson's question was almost certainly 'no', because we are not even allowed to use the arithmetic operations. But \the provides a loop-hole, since \the\n expands to the digits of \n, and we can then operate on them. It took me several hours to produce a solution along these lines, which was ugly and slow (but it wasn't a waste of time, since I learnt a lot about TEX's expansion mechanisms in the process). However I did eventually think of a much neater way. We need some initial definitions (which don't overwrite any macros in Plain TEX):

* The TEX implementation I usually use has a save stack of 600 words (the distribution default) and a main memory with 65535 (the default is about 30000).

```
\def\]{}\def\0{\sts\p}\def\1{\sts*\p}
\def\2{\sts**\p}\def\3{\sts***\p}
\def\4{\sts****\p}\def\5{\sts*****\p}
\def\6{\sts******\p}\def\7{\sts*******\p}
\def\8{\sts********\p}
\def\9{\sts*********\p}\def\q{}
\def\sts#1\p#2\q#3{\csname#3\endcsname
    #1#2#2#2#2#2#2#2#2#2\q}
```

After these, to set \asts to contain just \n asterisks you just have to type

```
\xdef\asts{\expandafter\sts
    \expandafter\p\expandafter\q
    \the\n]}
```

I have deliberately left the above uncommented as I hope some readers will enjoy working out for themselves how the macros work. Note the use of \csname...\endcsname to provide a look-up table; a trick that every TEXhacker should know, though I used it here because I wanted to eliminate conditional commands since I regard them as "almost" primitive commands. The macro works for bigger \n than the previous one. I have used to it to produce 54250 asterisks. Furthermore it seems to be marginally faster on the local implementation. I shall be interested to see whether anyone can find a still faster macro!

Exercise for METAFONT hackers: Appendix D of *The METAFONTbook* begins with the problem of defining a macro containing exactly n asterisks. Rewrite the above bits of TEX in METAFONT to solve this as well.

◇ George Russell
Mathematics
Trinity College
Cambridge, England
GER11@uk.ac.cam.phx

TEX and Envelopes

Dimitri Vulis

I have revised and improved the LATEX envelope macros that I posted to TeXHAX some years ago. Using them may save money.

Why bar codes on envelopes and other USPS gossip

It is reported that recently the United States Postal Service board of governors approved the 27-cent "public automation rate" for first-class mail whose envelopes are pre-printed with a ZIP+4 code[1] and a Postnet code, the bar code often found in lower right corner of business reply and courtesy envelopes, saving 2 cents off the new 29-cent rate for first-class mail. In the past, organizations simultaneously mailing 10 pieces in the same ZIP code, or mailing 250 and even 500 pieces pre-sorted by ZIP code were given discounts; now the discount may extend to single letters.

The existing Post Office sorting machines read the bar code placed in the lower right corner of a letter-sized envelope, but the new wide-area scanners, to be installed in the spring of 1991, will read the bar code virtually anywhere on the envelope, and it will be possible to bar code larger letters, magazines, and catalogs — so called flats.

USPS optical scanners already generate Postnet bar codes while processing envelopes with address legible enough for the optical character reader (i.e., not handwritten), but the Post Office would prefer to deal with letters already with a Postnet bar code. USPS expects to save $40 to $80 million on every 1% of mail that is sent "pre-bar-coded", and it passes a part of that saving back to the senders.

When a letter without a Postnet code is processed by the Post Office, an attempt is first made to feed it to an optical character reader (OCR) machine; if it succeeds in reading the address, it attempts to look up the ZIP+4 code in a database, sprays the Postnet code on the envelope, and from then on the envelope is handled automatically by bar code sorters (BCSs) at several points, until it reaches the destination post office; only then does a letter carrier read the address once again. However the OCR machines are known to be very finicky and it's very difficult to print an address that will be reliably scanned. The OCR machines want the

[1] The system of 9-digit numeric codes developed by the United States Postal Service that identifies small groups of delivery addresses.