Macros

Writing numbers in words in TEX

Edward M. Reingold

We present TEX macros to write integers, even extremely large integers, in words according to the American English nomenclature [2, pp. 12 and 22–24], [3, p. 1549]; the method here is easily adapted to the British English nomenclature, or that of other languages. The imaginative nomenclatures of [1, pp. 14–15] or [4, pp. 311–312] are also easy to accommodate with the ideas presented here. Although macros for writing numbers in words are already available on CTAN, none of them has the generality of those presented here. Our approach, which covers the full range of American English, $(-10^{66}, 10^{66})$, is based on [6, sec. 8.1]; [5, p. 6] has a similar method.

We want to be able to capitalize the first word produced, as well as insert spaces, commas, and hyphens between words appropriately. Because the words produced are written by various macros and at various levels of recursion, we centralize the production of text by calling a macro `\@String` that does the actual insertion of the word into the output. We use global flags to indicate whether the next word produced will be the first word (which should not be preceded by a space and which may need to be capitalized),

```
\def\@firstwordtrue{%
   \global\let\if@firstword\iftrue}
\def\@firstwordfalse{%
   \global\let\if@firstword\iffalse}
```

and similar global variables to indicate whether a capital letter is needed,

```
\def\@capitalfirstwordtrue{%
   \global\let\if@capitalfirstword\iftrue}
\def\@capitalfirstwordfalse{%
   \global\let\if@capitalfirstword\iffalse}
\@capitalfirstwordfalse
```

or whether a hyphen or comma is needed,

```
\def\@needhyphentrue{%
   \global\let\if@needhyphen\iftrue}
\def\@needhyphenfalse{%
   \global\let\if@needhyphen\iffalse}
\def\@needcommatrue{%
   \global\let\if@needcomma\iftrue}
\def\@needcommafalse{%
   \global\let\if@needcomma\iffalse}
\@needcommafalse
```

The macro that inserts a word into the output then is

```
\def\@String#1{%
  \if@firstword
    \expandafter\@firstoftwo\else
    \expandafter\@secondoftwo\fi
    {\if@capitalfirstword
      \Capitalize{#1}%
      \@capitalfirstwordfalse
    \else
      {#1}%
    \fi}%
   {\if@needhyphen
      {-#1}%
    \else
      \if@needcomma
          {, #1}%
          \@needcommafalse
      \else
          { #1}%
      \fi
    \fi}%
  \@firstwordfalse}
```

where capitalization is done by

```
\def\Capitalize#1{%
  \edef\@tempa{#1}%
  \expandafter\@capitalize
  \expandafter{\@tempa}\@EndOfString}
\def\@capitalize#1{%
   \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@capitalize#1}}
\def\@@capitalize#1#2\@EndOfString{%
   \uppercase{#1}#2}
```

Numbers less than twenty are idiosyncratic, so we handle them with a case statement:

```
\def\Small@Number#1{% Less than 20
  \relax
  \ifcase#1
    \@String{zero}\or
    \@String{one}\or
    \@String{two}\or
    \@String{three}\or
    \@String{four}\or
    \@String{five}\or
    \@String{six}\or
    \@String{seven}\or
    \@String{eight}\or
    \@String{nine}\or
    \@String{ten}\or
    \@String{eleven}\or
    \@String{twelve}\or
    \@String{thirteen}\or
    \@String{fourteen}\or
    \@String{fifteen}\or
    \@String{sixteen}\or
    \@String{seventeen}\or
    \@String{eighteen}\or
    \@String{nineteen}%
```

```
  \else
    \errmessage{Small number out of range}
  \fi}
```

We need some scratch counters to do our work:

```
\newcount\@number
\newcount\@@number
\newcount\@@@number
\newcount\@millenary
```

We use `\Medium@Number` to handle numbers smaller than 1000:

```
\def\Medium@Number#1{% At most three digits
  \@number=#1\relax
  \ifnum\@number>99
    \@@number=\@number
    \divide\@@number by 100
    \Small@Number{\the\@@number}%
    \@String{hundred}%
    \multiply\@@number by 100
    \advance\@number by -\@@number
  \fi
  % \@number is now \number mod 100
  \ifnum\@number>19
    \@@number=\@number
    \divide\@@number by 10
    % \@@number is now the tens digit
    \@Decade{\the\@@number}%
    \multiply\@@number by 10
    \advance\@number by -\@@number
    % \@number is now the ones digit
    \@needhyphentrue
  \fi
  % \@number is now 19 or less
  \ifnum\@number>0
      \Small@Number{\the\@number}%
  \fi
  \@needhyphenfalse}
```

where the "decade" is written by

```
\def\@Decade#1{%
  \ifcase#1
    \errmessage{Decade out of range}\or
    \errmessage{Decade out of range}\or
    \@String{twenty}\or \@String{thirty}\or
    \@String{forty}\or \@String{fifty}\or
    \@String{sixty}\or \@String{seventy}\or
    \@String{eighty}\or \@String{ninety}%
  \else
    \errmessage{Decade out of range}
  \fi}
```

Some usage requires the word "and" after the word "hundred", especially for the rightmost three digits of a number (for example, "one hundred and twenty"); this would require another global variable and a slight modification of `\Medium@Number`.

Numbers with four or more digits are handled recursively. To express $n \times 1000^i$ in words, we

express $\lfloor n/1000 \rfloor \times 1000^{i+1}$ in words,

express n mod 1000 in words, and

write the name of 1000^i in words.

The last step, writing the name of 1000^i, is done with

```
\def\@Millenary#1{%
  \ifcase#1\or
    \@String{thousand}\or
    \@String{million}\or
    \@String{billion}\or
    \@String{trillion}\or
    \@String{quadrillion}\or
    \@String{quintillion}\or
    \@String{sextillion}\or
    \@String{septillion}\or
    \@String{octillion}\or
    \@String{nonillion}\or
    \@String{decillion}\or
    \@String{undecillion}\or
    \@String{duodecillion}\or
    \@String{tredecillion}\or
    \@String{quattuordecillion}\or
    \@String{quindecillion}\or
    \@String{sexdecillion}\or
    \@String{septendecillion}\or
    \@String{octodecillion}\or
    \@String{novemdecillion}\or
    \@String{vigintillion}%
  \else
    \errmessage{Number too large for words}
  \fi
  \ifnum#1>0 \@needcommatrue\fi}
```

A "vigintillion" (10^{63}) is as high as American nomenclature goes; this is far larger than a TEX counter can go, but we are aiming high! With `\@Millenary` we translate our recursive structure into

```
\def\Big@Number#1#2{%
  \@@number=#2\relax    % number to be written...
  \@millenary=#1\relax  % times this power of 1000
  \ifnum\@@number>0
    \@@@number=\@@number
    \divide\@@@number by 1000
    \begingroup% Preserve \@millenary value
      \advance\@millenary by 1
      \Big@Number{\the\@millenary}%
                 {\the\@@@number}%
    \endgroup
    \multiply\@@@number by 1000
    \advance\@@number by -\@@@number
   % \@@number is now #2 mod 1000
    \ifnum\@@number>0
       \Medium@Number{\the\@@number}%
       \@Millenary{#1}%
    \fi
  \fi}
```

Calling `\Big@Number` produces `#2` $\times 1000^{\texttt{\#1}}$ in words, so the initial call to `\Big@Number` should have

a first parameter of zero. Thus we write the public macros

```
\def\inwords#1{%
  \edef\@tempa{#1}%
  \expandafter\@inwords\expandafter{\@tempa}}
\def\Inwords#1{%
  \@capitalfirstwordtrue
  \edef\@tempa{#1}%
  \expandafter\@inwords\expandafter{\@tempa}%
  \@capitalfirstwordfalse}
```

where

```
\def\@inwords#1{%
  \@firstwordtrue
  \@needcommafalse
  \@needhyphenfalse
  \ifnum#1<0
    \@String{minus}%
    \@number=-#1\relax
    \Big@Number{0}{\the\@number}%
  \else\ifnum#1=0
    \Small@Number{0}%
  \else%
    \Big@Number{0}{#1}%
  \fi\fi}
```

For example, `\inwords{-1234567890}` produces

> minus one billion, two hundred thirty-four million, five hundred sixty-seven thousand, eight hundred ninety

and `\Inwords{31415926}` produces

> Thirty-one million, four hundred fifteen thousand, nine hundred twenty-six.

The size limitation of TEX count registers, $2^{31}-1$, means that we get an error in trying to write 8018018851 in words (Conway and Guy [1, p. 15] call this "Knuth's number", the first prime number in the alphabetic ordering of the natural numbers [5, p. 4]). To write larger numbers in words we need to use the recursive structure of `\Big@Number` without resorting to count registers. This means that we have to trap a minus sign and ignore leading zeros, before we can split the number into the rightmost three digits and everything to their left. Hence we redefine

```
\def\Inwords#1{%
  \@capitalfirstwordtrue
  \inwords{#1}%
  \@capitalfirstwordfalse}
\def\inwords#1{%
  \@firstwordtrue
  \@needhyphenfalse
  \Trap@Minus{#1}}%
```

where

```
\def\Trap@Minus#1{%
  \edef\@tempa{#1}%
  \expandafter\@Trap@Minus%
  \expandafter{\@tempa}\@EndOfString}
\def\@Trap@Minus#1{%
  \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@Trap@Minus#1}}
\def\@@Trap@Minus#1#2{%
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\ifx#1-%
        \errmessage{Orphan minus sign}%
      \else
        \Small@Number{#1}\fi}%
     {\@@@Trap@Minus#1#2}}
\def\@@@Trap@Minus#1#2\@EndOfString{%
   \ifx#1-%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@String{minus}%
      \@TrapLeadingZeros{#2\@EndOfString}}%
     {\@TrapLeadingZeros{#1#2\@EndOfString}}}
```

traps a leading minus sign and then traps leading zeros with the similar

```
\def\@TrapLeadingZeros#1{%
   \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@TrapLeadingZeros#1}}
\def\@@TrapLeadingZeros#1#2{%
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\Small@Number{#1}}%
     {\@@@TrapLeadingZeros#1#2}}
\def\@@@TrapLeadingZeros#1#2\@EndOfString{%
   \ifx0#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@TrapLeadingZeros{#2\@EndOfString}}%
     {\@numberinwords{0}{#1#2\@EndOfString}}}
```

before calling a version of `\Big@Number` that avoids the use of count registers by splitting a number into the rightmost three digits and everything else:

```
\def\@numberinwords#1#2{%
   % #1 = power of 1000
   % #2 = the next token,  either
   %      a digit or \@EndOfString
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
   {}{\@@numberinwords{#1}{}#2}}
\def\@@numberinwords#1#2#3#4{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final one, #3
```

```
   % #3 = the next digit
   % #4 = the next token, either another
   %      digit or \@EndOfString
   \ifx\@EndOfString#4%
      \expandafter\@firstoftwo\else
      \expandafter\@secondoftwo\fi
      {\Small@Number{#3}\@Millenary{#1}#2}%
      {\@@@numberinwords{#1}{#2}#3#4}}
\def\@@@numberinwords#1#2#3#4#5{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final two, #3#4
   % #3#4 = the final two digits so far
   % #5 =  the next token, either another
   %       digit or \@EndOfString
   \ifx\@EndOfString#5%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\Medium@Number{#3#4}\@Millenary{#1}#2}%
     {\@@@@numberinwords{#1}{#2}#3#4#5}}
\def\@@@@numberinwords#1#2#3#4#5#6{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final three, #3#4#5
   % #3#4#5 = the final three digits so far
   % #6 = the next token, either another
   %      digit or \@EndOfString
   \ifx\@EndOfString#6%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@millenary=#1\relax
      \advance\@millenary by 1
      \@numberinwords{\the\@millenary}
                     {#2\@EndOfString}%
      \advance\@millenary by -1
      \ifnum#3#4#5>0
        \Medium@Number{#3#4#5}%
        \@Millenary{#1}%
      \fi}%
     {\@@@@numberinwords{#1}{#2#3}#4#5#6}}
```

Given this machinery, we can write Knuth's number using `\inwords{8018018851}`, eight billion, eighteen million, eighteen thousand, eight hundred fifty-one, and $2\times10^{63}+2\times10^{36}+2\times10^{12}+2293$, the last prime in alphabetical order [5, p. 12], two vigintillion, two undecillion, two trillion, two thousand, two hundred ninety-three. Or,

$$2^{219} = 842498333348457493583344221469363 \\ 458551160763204392890034487820288,$$

which in words is

> Eight hundred forty-two vigintillion, four hundred ninety-eight novemdecillion, three hundred thirty-three octodecillion, three hundred forty-eight septendecillion, four hundred fifty-seven sexdecillion, four hundred ninety-three quindecillion, five hundred eighty-three quattuordecillion, three hundred forty-four tredecillion, two hundred twenty-one duodecillion, four hundred sixty-nine undecillion, three hundred sixty-three decillion, four hundred fifty-eight nonillion, five hundred fifty-one octillion, one hundred sixty septillion, seven hundred sixty-three sextillion, two hundred four quintillion, three hundred ninety-two quadrillion, eight hundred ninety trillion, thirty-four billion, four hundred eighty-seven million, eight hundred twenty thousand, two hundred eighty-eight.

A final note: to number pages in words in LaTeX we need to `\protect` the call, as in:

```
\renewcommand*{\thepage}
      {Page \protect\inwords{\c@page}}
```

Acknowledgments The author is grateful to Nachum Dershowitz for pointing out various errors in the original macros, and to both him and Peter Wilson for suggesting the inclusion of appropriate hyphens and commas.

References

[1] John H. Conway and Richard Guy. *The Book of Numbers*. Springer-Verlag, New York, 1996.

[2] Philip J. Davis. *The Lore of Large Numbers*. Yale University, New Haven, CT, 1961.

[3] Philip B. Gove. *Webster's Third New International Dictionary of the English Language*. Merriam, Springfield, MA, 1961.

[4] Donald E. Knuth. Supernatural numbers. In David A. Klarner, editor, *The Mathematical Gardner*, pages 310–325. Wadsworth, Boston, 1981.

[5] Donald E. Knuth and Allan A. Miller. A programming and problem-solving seminar. Technical Report STAN-CS-81-863, Department of Computer Science, Stanford University, Stanford, CA, June 1981.

[6] Edward M. Reingold and Ruth N. Reingold. *PascAlgorithms*. Scott, Foresman and Company, Glenview, Illinois, 1988.

◇ Edward M. Reingold
Department of Computer Science
Illinois Institute of Technology
10 West 31st Street
Chicago, Illinois 60616-2987
USA
reingold (at) iit dot edu