Tutorial

✔

Fun with \if

Jonathan Fine

It is not commonly realised that the primitive \if commands of TEX (see [209], i.e. page 209 of *The TEXbook*) can be used within \message commands, when TEX is looking for a number or dimension, or an unexpandable token, and so forth. This article give some examples to illustrate the capabilities that are sometimes overlooked.

It is the author's custom to avoid all problems regarding unwanted space tokens in his macros by first making some category code changes

```
\catcode`\~=10    % space
\catcode`\ = 9    % ignore
\catcode`\^^M=9   % <return>
\catcode`\^^I=9   % <tab>
```

so now we use a '~' when a space is required. Note that [47] the '~' will produce a normal space character when read by the eyes of TEX.

Now for our first example. Suppose we have

```
\newif\ifsucceed
```

and we wish to \message the current value of \ifsucceed. We can write

```
\message
{
  \ifsucceed
    SUCCESS
  \else
    FAILURE
  \fi
}
```

rather than

```
\ifsuccess
   \message{ SUCCESS }
\else
   \message{ FAILURE }
\fi
```

which is slightly less efficient. The benefit of putting the \if within the \message rather than vice-versa is greater, the more text the two messages have in common.

Now suppose that we wish to use \read to compare two files for equality. One section of the code should read a line from each file, and compare them for equality. Before iterating, we should then test to see if one or both of the files has come to an end. Here then is the problem. We have \read streams \isA and \isB, and we wish to switch depending on the \ifeof values of \isA and \isB. We would like to avoid code such as

```
\ifeof \isA
  \ifeof \isB
    . . .
  \else
    . . .
  \fi
\else
  \ifeof \isB
    . . .
  \else
    . . .
  \fi
\fi
```

by using something similar to the AND logical operator.

Now think of this. The \if command results in expansion until two unexpandable tokens are found [209]. So we could arrange that \ifeof\isX produces a letter, and the two letters can then be compared. For example the code fragment

```
\if
  \ifeof \isA 0 \else 1 \fi
  \ifeof \isB 0 \else 1 \fi
```

behaves rather like the desired

```
( \ifeof\isA )  AND  ( \ifeof\isB )
```

which is not available to us.

It is possible to build up more complicated logical expressions in a similar manner. For simplicity, suppose the \ifA, ..., \ifZ are boolean quantities, created by \newif and manipulated through the \Xtrue and \Xfalse commands.

We want code that will test for one or more of the \ifX expressions being true. First form the expression

```
\if 0
      \ifA 1 \fi
      \ifB 1 \fi
      . . .
      \ifZ 1 \fi
    0
```

and think. The \if command will get the unexpandable 0 and will then look for the next unexpandable token, expanding the following macros and commands until it finds one. This will either be a 1, if at least one of the \ifX is true, or the trailing 0. In the first case the \if resolves to false, and the remainder of the expression is skipped. In the latter case, the whole expression has been absorbed by the \if, which is resolving to true. Thus,

```
NOT ( \ifA OR \ifB  . . . OR \ifZ )
```

is the analogue of the above expression.

To produce AND nesting rather than iteration is used. The arrow like formula

```
\if 0
      \ifA                 %   here
        . . .            %  is the
          \ifZ         %  point of
                1    %  the arrow  -
          \fi          %  how can
        . . .            %  we get
      \fi                    % here
    0
```

will resolve to false is all of the \ifX are true, and true otherwise.

Interesting though these constructions are, it is not clear that they are relevant to real-world problems found writing TEX macros. Therefore, I suggest the following challenge, which is simple enough to illustrate general principles, but complex enough to present genuine difficulties. It also has a real-world flavour. The task is to write macros which will compare two files for equality. I will present my own solution in a later article, unless of course I am sent a solution which is equal to or better than my own.

Have fun. Don't forget to turn the category codes back to there normal values.

MEX v. 1.05

Najnowsza wersja pakietu MEX ma numer 1.05 i datowana jest na 18 grudnia 1993 roku. Najważniejsza zmiana to wyeliminowanie błędu, wskutek którego niektóre znaki miały przypisaną nieprawidłową kategorię, mianowicie zbyt wiele znaków miało kategorię „litera” (\catcode=11); powodowało to m. in., że w akcentowanym słowie r\^^Dole sygnalizowany był błąd nieznany symbol kontrolny \^^Dole, podczas gdy wszystko byłoby dobrze, gdyby znak ^^D nie miał kategorii „litera”; obecnie (na początku pracy MEX-a lub LAMEX-a) znaki o kodach 0–127 mają kategorie dokładnie takie same, jak w odpowiednich formatach angielskich, natomiast wśród znaków o kodach powyżej 127 obowiązuje następująca zasada: znaki o kodach odpowiadających pozycjom polskich znaków diakrytycznych w układzie PL mają kategorię „litera”, pozostałe znaki mają kategorię „inny” (\catcode=12);

Makra

Szybki skład treści programów i dokumentacji

B. Jackowski i St. Wawrykiewicz

Bardzo często zachodzi potrzeba utworzenia szybkiego wydruku tzw. „listingu” programu bądź dokumentacji. Wydruk za pomocą systemowych poleceń copy *lub* type *pozbawia nas numeracji stron i informacji o nazwie pliku. Marginesy i font użyty przez drukarkę wymagają manipulacji na panelu sterowania lub umieszczania kodów sterujących bezpośrednio w pliku. Dzięki TEX-owi powyższe problemy odpadają pozwalając w sposób zautomatyzowany uzyskać w pełni użyteczne, czytelne wydruki.*

Przedstawione niżej makra oparte są na idei składu oddającego dosłownie zapis (tzw. verbatim) przedstawionej w *The TEXbook* (str. 380–382 i 421) oraz modyfikacjach B. Jackowskiego.

Skład dosłownego zapisu wymaga dostępu do znaków mających w TEX-u specjalne znaczenie. W tym celu musimy zmienić ich standardowe kody kategorii:

```
\def\uncatcode{%
 \catcode`\\12 \catcode`\{12
 \catcode`\}12 \catcode`\~12
 \catcode`\^12 \catcode`\%12
 \catcode`\#12 \catcode`\&12
 \catcode`\_12 \catcode`\$12 }
```

Ponieważ znaki TAB są w TEX-u zamieniane na spację, przyjmijmy całkiem arbitralnie, że w składzie verbatim rozwiną się one do czterech znaków spacji. Znak TAB musimy najpierw uaktywnić, nadając mu kod 13.

```
{\catcode`\^^I=13 \gdef^^I{\ \ \ \ }}
```

Jak widać kod może być poprzedzony nieobowiązkowym znakiem =.

Skład `verbatim` wykonujemy fontem `\tt`, którego wszystkie znaki (łącznie ze spacją) mają jednakową szerokość. Niestety w tym foncie zdefiniowane są dwie ligatury tzw. „hiszpańskie" — zapis sekwencji `!`` i `?`` daje w składzie pojedyncze znaki, odpowiednio: ¡ i ¿. Do naszych celów musimy zakazać tworzenia niepożądanych ligatur:

```
{\catcode`\`13 \gdef`{\relax\lq}}
```

Proszę zauważyć, że do zdefiniowania wyżej znaków TAB i ` musieliśmy lokalnie je uaktywnić. Ponowne ich uaktywnienie powinno nastąpić w ramach właściwego makra wykonującego całą robotę:

```
\def\listfile#1{{\tt \uncatcode
  \catcode`\^^I13  \catcode`\`13
  \def\par{\endgraf\leavevmode}
  \parindent0pt
  \obeylines\obeyspaces\input #1 }}
```

Komentarza wymaga przedefiniowanie komendy `\par`. Jak wiadomo makro `\obeylines` wstawia na końcu każdej linii komendę `\par`. Następuje zmiana trybu pracy TEX-a na pionowy. `\endgraf` jest równoważnikiem komendy `\par` zaś użycie `\leavevmode` wymusza natychmiast potem zmianę trybu na poziomy. Pozwala to na prawidłowe zadziałanie makra `\obeyspaces` odpowiedzialnego za odwzorowanie każdej spacji w składzie — ewentualne spacje ukażą się również na początku wierszy.

W nagłówku każdej strony powinna pojawić się nazwa pliku. Najprościej zrealizujemy to przez przedefiniowanie makra `\headline`.

```
\headline{\hfill\tt\N}
```

Co to jest `\N` zdefiniujemy sobie każdorazowo w ramach uruchomienia TEX-a. Jak można się domyślić będzie to nazwa pliku do „wylistowania". Bardziej rozbudowany nagłówek może zawierać pobrane z zegara systemowego dane o bieżącym czasie i dacie. Wykorzystamy tu TEX-owe polecenia pierwotne: `\time`, `\day`, `\month` i `\year` (*The TEXbook* str. 273, 349) i zmodyfikowane makra opublikowane w *The* METAFONT*book* str. 337 (dostępne również w pliku `testfont.tex`).

```
\newcount\godziny \newcount\minuty
\def\czas{% format `godz:min'
\godziny=\time \divide \godziny by 60 %
\minuty=-\godziny \multiply\minuty by60
\advance \minuty by \time
\ifnum\godziny>9 \the\godziny
       \else 0\the\godziny \fi :%
\ifnum\minuty>9 \the\minuty
       \else 0\the\minuty \fi}
%--------------------------------
\def\nazwamiesiąca{%
\ifcase \month%
\or stycznia\or lutego\or marca%
 \or kwietnia\or maja\or czerwca%
 \or lipca\or sierpnia\or września%
 \or października\or listopada%
 \or grudnia\fi}
%--- Modyfikacja \headline:
\headline{%
\it\czas\quad\the\day\ \nazwamiesiąca\
\the\year\ r.\hfill \tt\N}
```

Przekazanie parametru dla makra `\listfile` odbywa się w sposób trywialny kończąc formułowanie naszych makr, które zapiszemy w pliku o nazwie, np. `lst.tex`:

```
\listfile\N
\end
```

Uruchamiamy TEX-a za pomocą pliku wsadowego, nazwanego `list.bat`:

```
call mex \def\N{%1}\input lst
```

gdzie `mex` jest zależnym od indywidualnej konfiguracji wywołaniem TEX-a z formatem MEX (należy pamiętać aby `mex.bat` zawierał możliwość podania kilku parametrów!) zaś `%1` parametrem wywołania — nazwą pliku do listowania.

Napisany przez nas `list.bat` posiada niestety pewne wady. Otóż systemy operacyjne dopuszczają w nazwach plików znaki `_ % $ ^ & #` itp. Aby takie znaki pojawiły się również w nagłówku naszego wydruku musimy zmodyfikować wywołanie (ze względu na szerokość kolumny wypisane tu w trzech liniach):

```
call mex \let\C\catcode{\C`_12
   \C`^12\C`#12\C`&12\C`\%%12
   \C`$12\gdef\N{%1}}\input lst
```

Zapis został maksymalnie skrócony ze względu na ograniczenie długości lini poleceń systemu DOS do 127 znaków. Pozostaje problem do rozwiązania przez Czytelników: jak listować pliki zawierające w nazwie znaki { lub }.