Prolegomena toward a font selection scheme

Victor Eijkhout

Department of Computer Science University of Tennessee
104 Ayres Hall
Knoxville, Tennessee 37996, USA

eijkhout@cs.utk.edu

1 Introduction

Most users of plain TEX do not get very sophisticated in their use of fonts. Often they resort to declaring all used fonts explicitly with `\font`. There are some obvious disadvantages to that: it is not possible to switch a whole document in a simple way to a different typeface, or to a different size. As a result, I've seen such phenomena as an article with an abstract in 8 or 9 point, but where the formulas were still in 10 point, or pages of 'magnified' type where the lines were cramped, because the `\baselineskip` was not increased with the type size.

The need for a good font selection scheme is thus quite obvious, but the implementation of one is not.

2 Concepts

2.1 Font parameters

One point that most people agree on is that fonts are characterized by a number of parameters, and that it should be possible in an easy way to select the font that arises from changing just one of those parameters. For instance, switch to italics for emphasis, switch to a smaller size for a footnote, switch to the Greek alphabet to give an erudite – and untranslated, of course – quote from Homer.

However, the exact number of parameters is a point of contention. Appendix E of TEX Users Group, [2], gives macros that are based on size and style (`\rm`, `\it`, et cetera). An obvious third parameter – which Knuth, working exclusively in Computer Modern did not need – is the typeface. In [3, 4] Frank Mittelbach and Rainer Schöpf use an extra parameter, splitting style into 'shape' and 'series'. The series parameter combines the typographical qualities weight and width.

Furthermore, Karl Berry in [1] splits the typeface parameter into 'foundry' and 'typeface family'[1], and Yannis Haralambous suggested that the parameter 'alphabet' is also needed.

An international standard for font properties even exists: in section 8.6 of [5] 23 font properties are given. However, not all of these can be independently varied. For instance, the design copyright and the list of excluded characters are merely informational. The parameters suitable for inclusion in a font selection scheme are here: typeface name, posture (upright, oblique, italic, ...), weight (light, medium, bold, ...), proportionate width (condensed, medium, expanded, ...), structure (solid, outline, ...), and design size. Notably missing is the alphabet parameter.

2.2 More parameters, less parameters; do we have an argument?

To a certain degree, the above font classifications clash less than they seem to. Most parameters are orthogonal: if parameter 1 has p_1 values and parameter 2 has p_2 values, then all $p_1 \times p_2$ values make sense. But in that case we can introduce a new parameter 3 which has that many values. Thus the above schemes differ mostly in how far they split up the variety of fonts, not along what lines.

The number of parameters is to some extent a matter of taste, but a few common sense observations can be made.

- It is possible to live completely without any parameters by using only the control sequences that have been defined with `\font` declarations. This is very inconvenient.
- Certain DTP packages make it easy for you to (produce horrible typography: you simply) switch to 'outline' or 'shadow' (or both) for whatever font you are currently using. This is no reason to introduce independent parameters 'outline' and 'shadow' for every conceivable mutilation of typefaces.
- Some parameters are not likely to get varied: it is hard to conceive how the same font will be used from two different foundries in the same text, other than in texts that write specifically about typeface comparisons.
- Readers may have wondered why the distinction between serif and sans serif has not come up yet. The reason is that this is not an orthogonal category. Most typefaces exist only in seriffed or serifless

[1] It should be noted that this is not a proposal for of a font selection scheme, but for systematic filenames. The number of user parameters can either be more or less than the properties encoded in the file names.

design, but not both. For the occasional exception (Computer Modern, for instance) it is easiest to pretend that the seriffed and serifless subfamilies are simply two typeface families.

3 Implementation

The previous section described some of the issues involved in providing the user with suitable parameters for handling fonts. We must now look at the other side of the question: the parameters have to be translated into the name of a `tfm` file.

Any implementation of a font selection scheme has to pay attention to a couple of points.

- Certain combinations of parameters may not correspond to an existing font. A substitution may have to be performed, and in any case the user has to be told.
- For large sizes there must be a mechanism that makes a reasonable guess as to the file name.
- The memory usage of the font selection scheme should be as low as possible.

The last point can be elaborated a bit: memory usage should increase in a controlled manner if more parameters or more typefaces are added. Adding parameters may seem a bit strange to do, but it makes sense for such parameters as 'structure': suppose all fonts so far have been solid, and suddenly a printer is bought that can make an outline from any given font. Is the amount of memory for the font selection scheme then increased by a small quantity, or is it doubled?

Specifying for every combination of all parameters what file name results (as is done in the font selection scheme in [3]; in order "to implement the four dimensional grid of fonts" it keeps "for every combination of font family/series/shape" a list of size/external name pairs) is in one sense the optimal solution: any system (or absence thereof) of file names can be accommodated. However, the space needed is proportional to the product of all possible values of all parameters.

On the other hand, if file names are systematic (as in the naming scheme in [1], and as in the Computer Modern typefaces and all Bitstream and PostScript typefaces as far as I've seen), then considerable savings are possible: it becomes possible to translate a few parameters jointly into a fragment of the file name, and concatenate such fragments into the full file name. This reduces storage to a higher power root of the original amount[2].

This approach of componentwise translation offers an additional advantage for the 'size' parameter: for PostScript fonts the size can be given procedurally, for instance specifying

```
size: at #1pt
```

thus taking a fixed amount of storage for an infinite number of values.

An entirely modular translation is probably not possible. For instance, the translation of shape and series into a file name fragment is most likely dependent on the typeface, although it may be the same for all typefaces from a certain foundry.

Another impediment to modular translation is the handling of special cases. For instance, for certain typefaces certain styles may not be available, or styles may not be available in all sizes (probably the smaller sizes). However, such exceptions can most likely still be treated in a procedural manner.

References

[1] Karl Berry. Filenames for fonts. *TUGboat*, 11:517–520, 1991.

[2] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984. reprinted with corrections 1989.

[3] Frank Mittelbach and Rainer Schöpf. A new font selection scheme for TEX macro packages. *TUGboat*, 10:222–238, 1989.

[4] Frank Mittelbach and Rainer Schöpf. The new font family selection – user interface to standard LATEX. *TUGboat*, 11:297–305, 1990.

[5] ISO standard. Information technology – Font information interchange – Part 1: Architecture; international standard ISO/IEC 9541-1, 1991.

[2] For instance, translating three parameters independently makes storage proportional to the sum of the values of the three instead of the product. Thus storage is proportional to the cube root of what it would be in a fully explicit scheme.

From observation to publication

Theo A. Jurriens

Kapteyn Astronomical Institute P.O. Box 800
9700 AV Groningen
The Netherlands

taj@rugr86.rug.nl

Abstract

This article describes the use of TEX in publishing observations of variable stars observed by Dutch amateur-astronomers. The observations are published in the journal "Variabilia" and in the so-called Reports. In the latter the observations, collected in several years, are published and submitted to the professional astronomer. It includes tables and light-curves: plot of the changing magnitude of the star versus time. In creating the light-curves: PICTEX is used. In preparing the files for PICTEX simple TEX-coding is used for manipulating the data.

1 The data

Each observation is characterized by the observed star, a date, the brightness of the star and a code representing the observer. To avoid calender problems the so-called Julian Date is used. For example the Julian Date of 1961, september 9, my date of birth, is equal to 2437552 .

Each observation is TEX- coded as followed:

`\obs #1.#2.#3.#4.#5.`

`#1` is the integer of the Julian Date, `#2` it's fraction. `#3` is the integer of the magnitude, `#4` it's fraction and `#5` is the observer-code. The latter is not used for plotting. TEX can calculate with integers so the observing date and magnitude, the apparent brightness of a star, are transformed to integers. In case of date it is simply the integer of the Julian date. The magnitude is expressed in units of 0.1 so the magnitude is $10 \times \#3 + \#4$.

For each star we have a number of observations. The file of observations processed by `plain`- TEX produces a file suitable for PICTEX. This `.plt` file contains all the necessary information for a plot: the axis are proper scaled and labeled and contains information about the star. See the example at page 119 Figure 1 shows an typical file ready for TEX.

The macros used to create the `.plt` file are given below. Part II neglect the fraction of the Julian Date and calculates the magnitude, in units of 0.1. Both are written to a tempory file with the extension `.obs`. In part II also the minimum and maximum values along the axis are estimated. These values are used in Part III to calculate the proper sizes of the graph. In Part III the final `.plt` file is created and can be processed by PICTEX.

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%%  PART I plot.tex     %%
%%%%%%%%%%%%%%%%%%%%%%%%%
\newcount\tempx    \newcount\tempy
\newcount\tempz    \newcount\maxx
\newcount\minx     \newcount\miny
\newcount\maxy     \newcount\numobs
\newwrite\plotfile \newwrite\obsfile
\def\head#1{
\global\def\plotname{\sternum\ #1}
\initplot}
\def\endhead{\immediate\closeout\obsfile
\startplot}
\def\harvard#1{\global\def\sternum{#1}}
\def\type#1{\global\def\typename{#1}}
\def\initplot{
\immediate\openout\obsfile=\jobname.obs
\global\minx=99999
\global\maxx=0
\global\miny=99999
\global\maxy=0
\global\numobs=0}
\def\pplot#1#2{\tempx=#1
\tempy=\maxy
\tempz=#2
\advance\tempy by-\tempz
\advance\tempx by-\minx
\immediate\write\plotfile{\noexpand
\put {$\noexpand\bullet$} at
{\the\tempx} {\the\tempy}}}
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  PART II            %%
%%%%%%%%%%%%%%%%%%%%%%%%%%
\def\obs #1.#2.#3.#4.#5.{
\message{#5}
\global\advance\numobs by 1
\tempx=#1
% estimate min max x
\ifnum\minx>\tempx \global\minx=\tempx \fi
\ifnum\maxx<\tempx \global\maxx=\tempx \fi
\tempy=#3
\tempz=#4
% calculating magnitude
\multiply\tempy by 10
\advance\tempy by\tempz
% estimate min max y
\ifnum\miny>\tempy \global\miny=\tempy \fi
\ifnum\maxy<\tempy \global\maxy=\tempy \fi
\immediate\write\obsfile{\noexpand
\pplot{\the\tempx}{\the\tempy}}}
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  PART III           %%
%%%%%%%%%%%%%%%%%%%%%%%%%%
\def\startplot{
% create nice numbers along axis
\global\divide\minx by 10
\global\multiply\minx by 10
\global\divide\maxx by 10
\global\multiply\maxx by 10
\global\advance\maxx by 10
\global\divide\miny by 10
\global\multiply\miny by 10
\global\divide\maxy by 10
\global\multiply\maxy by 10
\global\advance\maxy by 10
\global\advance\maxx by-\minx
\ifnum\numobs>15
% writing pictex commands
 \def\name{\sternum.plt}
 \immediate\openout\plotfile=\name
 \message{\name}
 \immediate\write\plotfile{\noexpand
 \beginpicture}
 \immediate\write\plotfile{\noexpand
 \setcoordinatesystem units <0.35mm,1mm> }
 \tempz=\maxy
 \advance\tempz by-\miny
 \immediate\write\plotfile{\noexpand
 \setplotarea x from 0 to {\the\maxx},
 y from 0 to {\the\tempz}}
 \immediate\write\plotfile{\noexpand
 \plotheading{\plotname\ \typename}}
 \immediate\write\plotfile{\noexpand
 \axis bottom label {JD-{\the\minx}}}
 \immediate\write\plotfile{ticks
 numbered from 0 to {\the\maxx} by 40 / }
 \immediate\write\plotfile{\noexpand
 \axis left label {} }
 \immediate\write\plotfile{ticks
 from 0 to {\the\tempz} by 10 / }
 \tempx=\miny
 \advance\tempx by-10
 \tempz=\maxy
 \loop\ifnum\tempz>\tempx
        \tempy=\maxy
        \advance\tempy by-\tempz
        \immediate\write\plotfile{\noexpand
        \put {\the\tempz} at
        -15 {\the\tempy}}
         \advance\tempz by-10
 \repeat
% find out where to put text
% along y-axis.
 \tempz=\maxy
 \advance\tempz by-\miny
 \divide\tempz by2
 \tempx=\tempz
 \divide\tempx by10
 \tempy=\tempz
 \multiply\tempx by10
 \advance\tempy by-\tempx
 \ifnum\tempy=0
        \advance\tempz by5
 \fi
 \immediate\write\plotfile{\noexpand
 \put {Magn.} at -15 {\the\tempz}}
 \input \jobname.obs
 \immediate\write\plotfile{\noexpand\endpicture}
 \immediate\closeout\plotfile
\fi}
```

```
\input plot.tex
\harvard{000451}
\head{SS Cas}
\type{Mira  HIP}
\obs 47537.4.10.0.CMG.
\obs 47539.4.9.7.FJH.
\obs 47544.3.10.2.CMG.
\obs 47544.3.10.4.BMU.
\obs 47549.4.10.5.BMU.
\obs 47550.4.10.3.CMG.
\obs 47554.4.10.5.NWL.
\obs 47565.3.11.3.CMG.
\obs 47565.3.11.4.BMU.
\obs 47567.3.11.5.FJH.
\obs 47569.3.11.7.BMU.
\obs 47573.3.11.7.CMG.
\obs 47574.4.12.0.BMU.
\obs 47578.3.12.1.CMG.
\obs 47579.3.11.8.FJH.
\obs 47579.3.12.1.BMU.
\obs 47586.4.12.3.BMU.
\obs 47592.3.12.6.CMG.
\obs 47594.3.12.5.FJH.
\endhead
\bye
```

Fig. 1: *An input-file for TEX to create a file ready for PICTEX.*

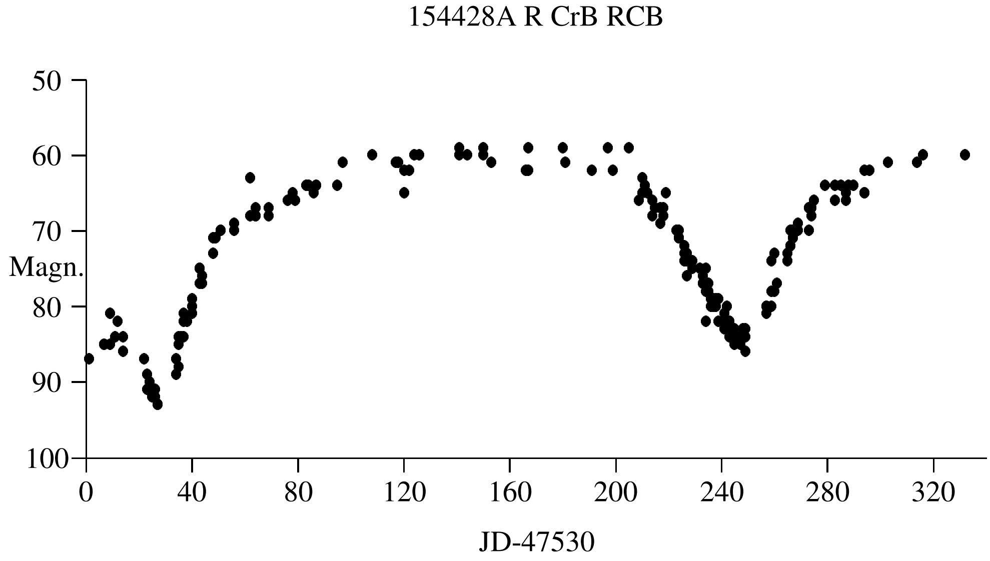


Fig. 2: *The light-curve of the star R CrB. The decrease in brightness can be seen clearly in this picture. The data are collected by Dutch amateur-astronomers.*

2 Macro's

In Part I the initialization is done: counters and files are defined. Information about the stars are stored in the tokens `\plotname`, `\sternum` and `\typename`. Also some initial values for the minimum and maximum values along the axis are set. The macro `\pplot` is used in Part II and executed in Part III. The macro writes the re-scaled X and Y values to the `.plt` file. Again the latter is used with PICTEX to obtain the desired result.

In Part II `\obs` is defined: the magnitude and Julian Date are converted to integers as described above and the maximum and minimum values are estimated according to the simple algorithme:
ifx_i < min then min := x_i;
ifx_i > max then max := x_i;
Also the number of observations is counted. This number is stored in `\numobs`.

In Part III the final plotting commands are written to the `.plt` file. This part is only executed if the number of observations is bigger than 15. The minima and maxima found in Part II are re-calcuted to have nice numbers along the axis. Commands for labels and numbering axis are also written into the file. After setting up the graph the data to be plotted are read from the `.obs` file. This file is created in Part II. Figure 3 shows such a `.plt` file.

```
\beginpicture
\setcoordinatesystem units <0.35mm,1mm>
\setplotarea x from 0 to {340},
y from 0 to {50}
\plotheading {154428A\ R CrB\ RCB}
\axis bottom label {JD-{47530}}
ticks numbered from 0 to {340} by 40 /
\axis left label {}
ticks from 0 to {50} by 10 /
\put {100} at -15 {0}
\put {90} at -15 {10}
\put {80} at -15 {20}
\put {70} at -15 {30}
\put {60} at -15 {40}
\put {50} at -15 {50}
\put {Magn.} at -15 {25}
\put {$\bullet $} at {1} {13}
\put {$\bullet $} at {7} {15}

⋮

\endpicture
```

Fig. 3: *The start of a typical plot-file as created by Part III.*

3 Publishing of the data

All the observations of one year are collected in one file with a structure as in Figure 1. This file contains 6000 to 10.000 lines depending on the weather conditions. The file is processed using `\plot.tex` and a number of `.plt` files are created. The same file is used to create a six-column tabular output of the observations, see the example at the next page. These macro's are available on request.

table page to be inserted by hand

FIFO and LIFO incognito

Kees van der Laan

February 1992

Abstract

FIFO, first-in-first-out, and LIFO, last-in-last-out, are well-known techniques for handling sequences. In TeX macro writing they are abundant but are not easily recognized as such. TeX templates for FIFO and LIFO are given and their use illustrated.

1 Introduction

It started with the programming of the Tower of Hanoi in TeX, see van der Laan (1992). For printing each tower the general FIFO — First-In-First-Out[1] — approach was considered.[2] In literature (and courseware) the programming of these kind of things is done differently by each author, inhibiting intelligibility. In pursuit of Wirth (1976) TeX templates for the FIFO (and LIFO) paradigma will hopefully improve the situation.

2 FIFO

FIFO can be TeXed as template via[3]

```
\def\bfifo#1#2\efifo{\process{#1}
\ifx\empty#2\empty
\else\bfifo#2\efifo\fi
}%end \bfifo...\efifo
```

Printing of a tower  can be done via

```
\def\process#1{\kern.2ex\hbox to3ex{%
\hss\vrule width#1ex height1ex\hss}}
\vbox{\offinterlineskip\bfifo12\efifo}
```

The `\bfifo...\efifo` macro is a basic one. It allows to procede along a list and to apply a (user) specified process to each list element. By this approach the programming of walking through a list is separated from the various processes to be applied to the elements. Fundamental![4]

The recursion will be terminated if #2 is empty.[5] One can circumvent the building up of `\fi`'s via[6]

```
\def\bfifo#1#2\efifo{\process{#1}
   \ifx\empty#2\empty
   \else\def\aux{\bfifo#2\efifo}
        \expandafter\aux\fi
}%end \bfifo...\efifo
```

or via

```
\def\bfifo#1#2\efifo{\process{#1}
   \ifx\empty#2\empty\let\aux=\relax
   \else\def\aux{\bfifo#2\efifo}
   \fi\aux
}%end \bfifo...\efifo
```

A more TeX-like implementation is

```
\def\bfifo#1{%
   \ifx\efifo#1\let\nxt=\relax%
   \else\def\nxt{\process{#1}\bfifo}%
   \fi\nxt}%end \bfifo
```

The advantage of the last implementation is that the input stream is processed one group or token at a time until `\efifo` is encountered. Moreover, it can handle the invoke `\bfifo\efifo`, the empty case. No auxiliary stacks are involved. This way of programming is unusual for those familiar with PASCAL-like programming.

[1] See Knuth (1968), section 2.2.1.

[2] In the Tower of Hanoi article Knuth's list datastructure was finally used — TeXbook Appendix D.2 — with FIFO inherent.

[3] My first version had the two tokens after `\ifx` reversed — a cow flew by — and made me realize the non-commutativity of the arguments of TeX's conditionals. In math and in programming languages like PASCAL the equality relation is commutative! Note that at least one argument is needed in the above given implementation of FIFO.

[4] If a list has to be *created,* Knuth's list datastructure might be used, however, simplifying the execution of the list. See TeXbook Appendix D.2.

[5] Note that the second `\empty` is not always necessary. Knuth and Mackay (1987) demonstrate yet another variant of programming the test. The above given form is in agreement with Knuth's style as demonstrated in `\displaytest`, see the TeXbook, Appendix D-1, p.376.

[6] See Kabelschacht (1987).

2.1 Variable number of parameters

TEX macros can take at most 9 parameters. The above `\bfifo` macro can be seen as a macro which is relieved from that restriction. Every group or token in the input stream after `\bfifo` will become an argument to the macro. The first token or group is the first argument to the first invoke. This invoke ends with an invoke of itself using the next token or group from the input stream as argument. So the second token is argument of the second invoke. In general the n^{th} token or group is argument of the n^{th} invoke and so on until the `\efifo` token is reached, whereupon no invoke of `\bfifo` will occur.[7]

2.2 Length of string

An alternative to Knuth's macro, TB219, is obtained via an appropriate definition of `\process`.

```
\newcount\length
\def\process#1{\advance\length1}
```

Then `\bfifo aap\efifo` yields the length **3**.

2.3 Vertical printing

David Salomon treats the problem of vertical printing in his courseware. Via an appropriate definition of `\process` and a suitable invoke of `\bfifo..\efifo` it is easily obtained.

```
\def\process#1{\hbox{#1}}
xy\vbox{\bfifo abc\efifo}yx
```

yields xy$\begin{matrix}a\\b\\c\end{matrix}$yx.[8]

2.4 Delete last character of argument

Again an example due to David Salomon. It is related to the well-known `\gobble` macro to eat the *next* token (or group) from the input stream. One could define an appropriate `\process` but that will require double testing. Simpler is the following modification of the `\bfifo...\efifo` template.

```
\def\bgobblelast#1#2\egobblelast{
   \ifx\empty#2\empty\let\aux=\relax
   \else#1\def\aux{\bgobblelast#2%
    \egobblelast}\fi\aux%
}%end \bgobblelast...\egobblelast
```

Then `\bgobblelast aap\egobblelast` will yield `aa` .

2.5 To process words

In document preparation it is important to be able to handle quantities sequentially as elements of a list. What about handling a list of words? Amy Hendrickson in her courseware considers among others the problem of underlining words. This can be done by underlining every character, but that is slow. A faster solution can be obtained by first modifying the `\bfifo...\efifo` template into a version which picks up words, and to give `\process` the function to underline its parameter.[9]

```
\def\bfifow#1 #2\efifow{\process{#1}%
%Process words recursively;
%no addition of space here (part of
%\process if needed).
  \ifx\empty#2\empty\let\aux=\relax
  \else\def\aux{\bfifow#2\efifow}%
  \fi\aux}%end \bfifow...\efifow
```

The more TEX-like implementation, where the input stream is processed word wise, reads

```
\def\bfifow#1 {%
   \ifx\efifow#1\let\nxt\relax
   \else\def\nxt{\process{#1}\bfifow}
   \fi\nxt}%end \bfifow
```

2.5.1 Underlining words

In print it is unusual to emphasize words by underlining. Generally another font is used, see discussion of exercise 18.26 in the TEXbook. However, now and then people ask for (poor man's) underlining of words. The following `\process` definition underlines words picked up by `\bfifow...\efifow`.

```
\def\process#1{\vtop{\hbox{\strut#1}
                \hrule}\ }
\leavevmode\bfifow leentje leerde lotje
        lopen langs de lange lindenlaan
\efifow\unskip.
```

yields:[10]
leentje leerde lotje lopen langs de lange lindenlaan.

Note that underlining of complete sentences has to be considered separately if underlining punctuation marks is forbidden. (One possibility is to redefine `\process` such that the last symbol of its argument is inspected and appropriate action taken; another possibility is to use the general `\bfifo...\efifo` macro and suppress underlining for punctuation marks via appropriate programming of `\process`.)

3 Nested FIFO

One can nest the FIFO paradigma for example for processing lines word per word.[11] The template reads

```
\def\bfifol#1^^M#2\efifol{
  \bfifow#1\efifow
  \ifx\empty#2\empty\let\auxl=\relax
  \else\def\auxl{\bfifol#2\efifol}
  \fi\auxl}%end \bfifol...\efifol
```

with `\bfifow...\efifow` as defined above.

[7] Another way to circumvent the 9 parameters limitation is to associate names to the quatities to be used as parameters, let us say via def's, and use these quatities via their names in the macro. This is related to the so-called keyword parameter mechanism of command languages.

[8] Note the use of the `\hbox...` in process.

[9] Note that underlining inhibits hyphenation.

[10] `\unskip` is needed to undo the insertion of the last space.

[11] Or character per character, token per token, or group per group.

3.1 Natural data

Data for `\h(v)align` needs & and `\cr` marks. We can get plain TEX to insert an automatic `\cr` at each (natural) input line, TEXbook p.249. An extension of this is to get plain TEX to insert `\cs`-s, column separators, and `\rs`-s, row separators, and eventually to add `\lr`, last row, at the end, in natural data. For example prior to an invoke of `\halign`, one wants to get plain TEX to do the transformation

P*ON	$\Rightarrow$	P\cs*\csO\csN\rs
DEK*		D\csE\csK\cs*\lr

This can be done via adaptation of the above template along with an appropriate `\process` definition.

```
\let\ea=\expandafter
\newdimen\csize\csize=3ex
\def\rs{\cr}%generalization of row sep
\def\lr{\cr}%last row
\def\cs{&}%generalization of column sep
\catcode`*=13 \def*{%crossed out cell
\vrule width0.6\csize height0.5\csize %
depth0pt}%simple BLACK variant
%more pleasing is the following
%poor man's grey
\newbox\crs
\setbox\crs=\hbox to.6\csize{\leaders%
\hbox to.2ex{\hss\vrule height.5\csize
depth0pt\hss}\hfil}
\def*{\copy\crs}
%
\catcode`\^^M=13 \let^^M=\relax
%Pick up and processing of lines
\def\bfifol#1^^M#2\efifol{%
\process{#1}%
\ifx\empty#2\empty\def\auxl{\lr}%
\else\def\auxl{\rs\bfifol#2\efifol}%
\fi\auxl}%ensure end conditional before
         %inserting tabular mark up
\def\process{\bfifo#1\efifo}%
%
%Pick up etc of chars per line
\def\bfifo#1#2\efifo{#1%#1 back
\ifx\empty#2\empty\let\aux=\relax%
\else\cs%insert \cs
\def\aux{\bfifo#2\efifo}\fi%
\aux}%
```

To demonstrate that it wor—-hey it works!—ks

```
%%%%%%% data provision %%%%%%%
\def\data{%
P*ON
DEK*
}%
%%%%%%% data transform %%%%%%%
\ea\def\ea\data\ea{\ea%
\bfifol\data\efifol}%define transform
%%%%%%%   application   %%%%%%%
$$\vbox{\halign{&\hbox to\csize{%
\vrule height.8\csize width0pt
depth.2\csize\hfil#\hfil}\cr\data}}$$%
```

will yield

P   O N
D E K

As may be guessed from the layout the above came to mind when typesetting crosswords, while striving after the possibility to allow natural input, independent of `\halign` processing. Note that a weak form of the look ahead principle is implicitly applied as well.

4 LIFO

A modification of the `\bfifo...\efifo` macro—`\process{#1}` invoked at the end instead of at the beginning—will yield the Last-In-First-Out template. Of course LIFO can be applied to reversion 'on the flight,' without explicitly allocating auxiliary storage.[12]

```
\def\blifo#1#2\elifo{%
   \ifx\empty#2\empty\let\aux=\relax%
   \else\def\aux{\blifo#2\elifo}\fi%
   \aux\process{#1}%
}%end \blifo...\elifo
```

With the identity—`\def\process#1{#1}`—the template can be used for reversion. For example `\blifo aap\elifo` yields paa.

5 Further reading

Zalmstra and Rogers (1989), apply the FIFO technique to a list of figures — or floating bodies — in order to merge the list appropriately with the main vertical list in the output routine. This is beyond the scope of this paper.

6 Conclusion

In looking for a fundamental approach to process elements sequentially—not to confuse with list processing where the list is also built up, see TEXbook Appendix D.2—TEX templates for FIFO and LIFO, emerged.

The templates can be used for processing lines, words or characters. Also processing of words or characters per line can be handled via nested usage of the FIFO principle.

TEX's conditionals are non-commutative, while the similar mathematical and programming operations are.

From the application point of view the FIFO principle along with the look ahead mechanism is applied to molding natural data into representations required by subsequent TEX processing.

[12]Johannes Braams drew my attention to Knuth and MacKay (1987), which contained among others `\reflect...\tcelfer`. They compare #1 with `\empty`, which is nice. The invoke needs an extra token, `\empty` — a so-called sentinel, see Wirth (1976) — to be included before `\tcelfer`, however. (Knuth and Mackay hide this by another macro which invokes `\reflect...\empty\tcelfer`). My approach requires at least one argument, with the consequence that the empty case must be treated separately, or a sentinel must be appended after all.

References

[1] Hendrickson, A (priv. comm.)

[2] Kabelschacht, A (1987): `\expandafter` in conditionals; a generalization of plain's `\loop`. *TUGboat* 8, no. (2), 184–185.

[3] Knuth, D.E (1968): The Art of Computer Programming. 1. Fundamental Algorithms. Addison-Wesley.

[4] Knuth, D.E, P. Mackay (1987): Mixing right-to-left texts with left-to-right texts. *TUGboat* 7, no. (1), 14–25.

[5] Knuth, D.E (1984): The TEXbook. Addison-Wesley.

[6] Laan, C.G. van der (1992): Tower of Hanoi, revisited. *TUGboat* 13, no. (1), 91–94.

[7] Salomon, D (priv. comm.)

[8] Wirth, N (1976): Algorithms + Data Structures = Programs. Prentice-Hall.

[9] Zalmstra, J, D.F. Rogers (1989): A page make-up macro. *TUGboat* 10, no. (1), 73–81.

Tower of Hanoi, revisited

Kees van der Laan

December 1991

Abstract

Another version of programming 'The Tower of Hanoi' in TEX is provided.[1] No nodding knowledge of Lisp is required; just plain TEX. There is no restriction on the number of disks, apart from the installed limits of TEX. Generalized disks can be moved as well.

1 Introduction

At the Dedham TUG91 conference, I attended David Salomon's advanced TEX course. Instead of redoing his clear and ample exercises I decided to rework Leban(1985). The more so because elaborating a classic example might bring you to fundamental issues. In courses on programmming the Tower of Hanoi problem is used to illustrate paradigms. I was pleased to encounter some paradigms of TEX programming while revisiting the tower.

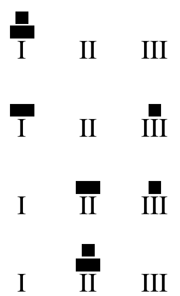
2 The Tower of Hanoi problem

A pyramid of disks—meaning a tower with implicit ordering of the disks—has to be moved under the restrictions that only one disk at a time can be moved and that each intermediate state consists of pyramids, obeying the original implicit ordering. In total three places for (intermediate) pyramids are allowed. For a pyramid of n disks, the solution needs $2^n - 1$ moves. For an introduction to the problem see the first paragraphs in Graham c.s.(1989).

3 Example

```
\input hanoi.tex \hanoi2 \bye
```

will yield the process of replacements for 2 disks from tower I to tower II



[1]For an earlier article on the issue, see Leban(1985).

4 The file hanoi.tex

This file contains all the macros: the top `\hanoi`, the version with more parameters `\Hanoi`, the macro for the moves `\movedisk`, along with the auxiliary macro to prefix a string, `\logoffx...\logoffx...`, and the auxiliaries for printing `\showtowers`, and `\pt`.

As data structure a simplified version of Knuth's list, see TEXbook Appendix D.2, is used. A tower has the replacement text $\backslash\backslash\langle\text{item}_1\rangle$ $\backslash\backslash\langle\text{item}_2\rangle$ ... $\backslash\backslash\langle\text{item}_n\rangle$, where in this case each item is a control sequence. The separators, \\, '... are enormously useful, because we can define \\ to be any desired one-argument macro and then we can *execute* the list!'

```
%hanoi.tex version 18 dec 91
\newcount\n   %The number of disks
\newcount\brd %Breadth of towers
\newcount\hgt %Height of maximum tower
\newcount\dskhgt %Height of each disk
\let\ea=\expandafter %Shorthand
\let\ag=\aftergroup  %Shorthand
\def\preloop{%To create loopcnt, a
             %local loopcounter
             %(see also loopy.TeX).
   \bgroup \advance\count10 by 1
   \countdef\loopcnt=\count10
            %Symbolic name
   \loopcnt=1 %(default)
 }%end \preloop
\def\postloop{\loopcnt=0 %Restore
   \egroup}%end \postloop
%
%Hanoi macros, top level
\def\hanoi#1{%Argument can be digit(s)
             %or a counter (numeric)
 \n=#1 %Assign argument value to \n
 \def\II{}\def\III{}%Empty towers
 %Next is inspired by the TeXbook,
 %p374, 378
 %The initial tower for \I is created
 %The initial tower for \I is created
```

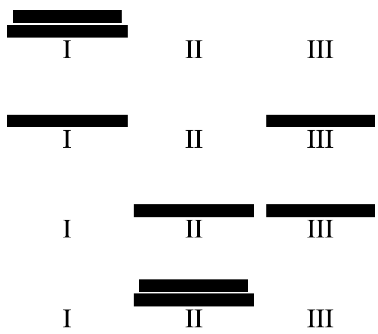
```
%   \def\I{\\\i\\\ii\\\iii...\\\'n'}
%next to the defs for \i,\ii,...\'n'.
\preloop\ag\def\ag\I\ag{%
 \loop
   \ea\xdef\csname\romannumeral\loopcnt
            \endcsname{\the\loopcnt}
   \ag\\%separator
   \ea\ag\csname
        \romannumeral\loopcnt\endcsname
   \ifnum\loopcnt<\n
   \advance\loopcnt by 1
 \repeat   \ag}
\postloop
%For printing, values are needed for
\brd=\n %Breadth of largest disk
\advance\brd by 3 %Little room extra
\dskhgt=1 %Height of disks
\hgt=\n\multiply\hgt by2 %\hgt is height
      \advance\hgt by1 %of towers
\showtowers %Print initial state
\Hanoi\I\II\III\n
}%end \hanoi
%
\def\Hanoi#1#2#3#4{%Moves from #1 to #2,
                   %with aid of tower #3.
 %The number of disks is #4, in a counter.
 \ifnum#4=1 %For Tower of 1 disk,
            %just move the disk
   \movedisk\from#1\to#2%
   \showtowers%Print towers after move
 \else%Problem of #4 disks is solved by
      %- problem of (#4-1) disks,
      %- a move, and
      %- a problem of (#4-1) disks.
   {\advance#4 by-1 \Hanoi#1#3#2#4}%
   \movedisk\from#1\to#2%
   \showtowers%Print towers after move
   {\advance#4 by-1 \Hanoi#3#2#1#4}%
 \fi}%end \Hanoi
%
%Moving of the disks, TeXbook, App. D.2
%Slightly adapted versions of \lop (
%called \movedisk with function that
%first element of #1 is prefixed to #2)
%and \lopoff modification
\def\movedisk\from#1\to#2{%Move disk from
%tower #1 to tower #2
  \ea\lopoffx#1\lopoffx#1#2}
\def\lopoffx\\#1#2\lopoffx#3#4{\ea\gdef%
  \ea#4\ea{\ea\\\ea#1#4}
  \gdef#3{#2}%restore stub}%end\lopoffx
 }%end \movedisk
%
%Printing tower status
\def\showtowers{%Display pyramids
 \par\quad\hbox{\pt\I\ \pt\II\ \pt\III
               }\par
 }%end \showtowers
%
%Auxiliaries
\def\gobble#1{}%To eat character
%
\def\\#1{\hbox to\brd ex{\hss%
  \vrule width#1ex height\dskhgt ex%
                          \hss}%
 }%end \\
%
\def\pt#1{%Print Tower.
%#1 is \I, \II, or \III
\vbox to\hgt ex{\baselineskip=.2ex\vss%
      #1%
      %Format pointer underneath
      \hbox to\brd ex{\hss%
              \ea\gobble\string#1\hss}%
      }%end vbox
 }%end \pt
```

5 Disks not restricted to one digit

One could invoke `\hanoi{10}`[2] at the expense of ample time and use of paper. In order to illustrate the possibility of the macros to cope properly with disks denoted by more than one digit, the pyramid $\Big/ {9 \atop 10} \Big\backslash$ can be handled via

```
\n=2    %Number of disks
\def\ix{9}\def\x{10}%Disks
\def\I{\\\ix\\\x}\def\II{}\def\III{}
\brd=10 %Breadth of largest disk
\hgt= 6 %Height of tower
\dskhgt=1 %Height of disks
\def\\#1{\hbox to\brd ex{\hss%
  \vrule width#1ex height\dskhgt ex%
                          \hss}}%
\showtowers%Initial state
\Hanoi\I\II\III\n
```

with result



[2] Or an invoke with an even larger argument. I allowed my modestly equipped MS-DOS PC to loop for $\backslash n = 1, 2, 3, \ldots$ in order to experience TEX's limits with respect to the practical upper bound for the size of the tower. A tower of 10 disks did take ≈ 1.5 minutes to TEX, with empty `\showtowers`. I had the patience to await the TEXing up to the tower of size 16, and concluded that there are no limits for reasonable usage. (Note that every increase of the size by 1 will double the time needed.) So, I don't know when my PC will crash or, to paraphrase the monks, 'when the world will end.' I also modified the macros into versions with `\I`, `\II`, and `\III` defined as token variables. I was surprised to see *that* version run 4 times slower. Whatever the value, the hanoi macros can be used as a program to compare the efficiency of TEX implementations.

6 Generalized disks

What about for example `(xyz)` as disk? Let us assume for printing that the contents of the pyramids—the strings—will do, in the implicit provided order. This can be obtained via a modified `\\` definition.

7 Example $\Big/\begin{matrix}\spadesuit\\ \textbf{(xyz)}\end{matrix}\Big\backslash$

The tower can be moved, with the states printed via

```
\n=2\def\i{$\spadesuit$}\def\xyz{(xyz)}
\def\I{\\\i\\\xyz}\def\II{}\def\III{}
\brd=6 \hgt=7
\def\\#1{\hbox to \brd ex{\hss%
                          #1\hss}}%
\showtowers%Initial state
\Hanoi\I\II\III\n
```

with results

♠ (xyz)		
I	II	III

(xyz)		♠
I	II	III

	(xyz)	♠
I	II	III

	♠ (xyz)	
I	II	III

8 Interactivity

Downes(1991) inspired me to think about direct communication with the user. What about the modification of `\showtowers` into an appropriate `\immediate\write16{...}` command, such that the moves will appear on the screen? No previewing nor printing![3]

```
%Direct screen \showtowers
\def\showtowers{
\immediate\write16{
   \ea\gobble\string\I:   \I
   \ea\gobble\string\II:  \II
   \ea\gobble\string\III: \III}}
```

9 Conclusion

Is this just for fun? It was appropriate for 'Fun with TEX,' NTG's 91 fall meeting at Eindhoven. Furthermore, I experienced the following fundamental (TEX) programming issues

- recursion in solving the problem
- the use of the list data structure and separators to execute the list, see the TEXbook Appendix D.2
- creating and using a local loop counter[4]
- creation of a dynamic number of command names and a string of dynamic length
- generalizing the problem (not only numbers can denote disks; note that no comparison of disks is done, the ordering of the subtowers is maintained implicitly)
- direct communication on screen.

TEXnically `\aftergroup`, `\countdef`, `\csname`, `\expandafter`, `\ifnum`, `\ifx`, `\immediate\write16...`, `\loop`, `\romannumeral`, and `\string` are exercised.

The hardest thing was to get the towers aligned when formatting commands were split over several lines, due to the two-column format. Several % symbols were needed to annihilate the effect of spurious spaces, especially those created by some ⟨cr⟩'s.

It did take some time to realize the benefits of Knuth's list macros, not to say that I wandered around quite a bit.

References

[1] Downes, M.J. (1991): Dialogue with TEX. Proceedings TUG91.

[2] Graham, R.L, D.E. Knuth, O. Pastashnik (1989): Concrete Mathematics. Addison-Wesley.

[3] Knuth, D.E. (1984): The TEXbook. Addison-Wesley.

[4] Leban, B. (1985): A solution to the Tower of Hanoi problem using TEX. *TUGboat* 6, no. (3), 151–154.

[5] Pittman, J.E. (1988): Loopy.TeX. *TUGboat* 9, no. (3), 289–291.

[6] Salomon, D. (priv.comm.)

[3] Alas, no control over the format on the screen either.

[4] This was mentioned before by Pittman(1988). It is a matter of taste and programming style whether one prefers this next best to the hidden counter idea, above the use of a global counter, for counting the number of times a loop is traversed. The difference in efficiency is negligible.

Typesetting Crosswords via TEX

Kees van der Laan

March 1992

Abstract

A macro is provided for typesetting crosswords via (plain) TEX, or any TEX, which allows `\halign` use. The specification of the crossword information can be done in the WYSIWYG way,[1] and does not require `\halign` markup, just the data.

Keywords

Crosswords, games, plain TEX.

1 Introduction

This work emerged from my work on tables and FIFO—First In First Out.[2]

Hamilton Kelley (1990) published sophisticated LATEX macros for ‘drawing’ crosswords.

Although not a crossworder myself, it comes to mind that typesetting crosswords comes down to

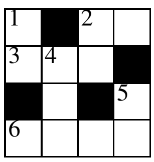
- specification of the puzzle and the solution
- providing the clues
- typesetting it all.

It is not difficult to use `\halign`. The difficulty is to keep it simple and flexible, to adopt simple conventions, and to allow natural input.

2 Examples

First the puzzle, next the clues and finally the solution to the puzzle are given as examples of use.

2.1 Puzzle



is obtained via

```
\input crw.tex
\bdata%
P*On
DEk*
*n*S
Edit
\edata\markup\data
$$\puzzletrue\crw\data$$
```

Conventions for `\bdata` ⟨ *data* ⟩ `\edata`[3]

- cell descriptions have to be given per line
- * denotes crossed out cell
- capitals denote marked open cells (with reference numbers to the clues), and letters of the solution
- lower case letters, denote empty cells, and letters of the solution.

I chose to provide the crossword information via the above given conventions, because it is natural and can be prepared independently. I refrained from the interactive provision of the information within TEX, because I don’t know how to correct typos easily in that way.

`\markup\data` inserts the mark up necessary for the expansion of `\crw`, via redefinition of `\data`.

`\puzzletrue\crw\data` typesets the puzzle.

2.2 Clues

Is there a problem? Just columns of text, eventually aligned.[4] This subsection has nothing to do with the file `crw.tex`.

---

[1] What You See Is What You Get

[2] Table Diversions, to be presented at EuroTEX92, and FIFO and LIFO incognito, submitted for publication.

[3] A ‘white lie,’ spaces are partly allowed for crossed out cells, see the Appendix.

[4] Admitted, it is not trivial to have the clues formatted with the numbers hanging out without mark up information in the text. This can be handled by a macro with two arguments and a space as separator to distinguish the clue number from the clue text. This macro can be invoked after lines have been taken apart via `\bfifol...\efifol`. It is analogous to the nesting applied in `\markup`. This can be suitably applied within a `\valign`.

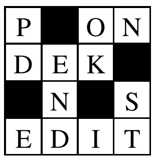
Across	Down
2 Switch mode	1 Public domain
3 Knuth	2 All right
6 Prior to TeX	4 All comes to it
	5 Atari type

is obtained via

```
\smallskip\noindent
\vtop{\hsize=21ex\obeylines Across
2 Switch mode
3 Knuth
6 Prior to \TeX
}\vtop{\hsize=25ex\obeylines Down
1 Public domain
2 All right
4 All comes to it
5 Atari type}
```

2.3 Solution

`\puzzlefalse` toggles typesetting the solution, in uppercase. After `\input crw.tex`, data definition and markup, the solution



is obtained via

```
$$\def\num{}\crw\data$$
```

Redefinition of `\num`—`\def\num{}`—suppresses the typesetting of the reference numbers for the clues.

3 Programming

Crossword diagrams can be characterized by a table with 1-character elements: (marked) empty cells, crossed out cells, and letters. A cell is `\csize` by `\csize`, with height `.8\csize`. The carriage return, `^^M`, is an active character allowing WYSIWYG input. The consequence is that no `&`-s nor `\cr`-s are needed in the data.[5] The nice side-effect is that visual verification of the input is alleviated. The WYSIWYG provided data is transformed into marked up data ready for use within `\halign`[6] by the macro `\markup`. An inefficiency is that the *programming* of the markup is done once, while the mark up will be *inserted* whenever `\data` is invoked. The macro `\crw` typesets the crossword. The numbering of the marked cells is done automatically, row-wise and hidden.

3.1 File crw.tex

```
\let\ea=\expandafter \newif\ifpuzzle
\newcount\cnt\newdimen\csize\csize=3ex
%
\def\crw#1{\toxit{\offinterlineskip%
\halign{\bcell width0pt%of strut
\prc##\ecell&&\bcell\relax
\prc##\ecell\cr#1}%end \halign
\global\cnt0}}%end \crw
%
\def\toxit#1{\vtop{\hrule
\hbox{\vrule\vbox{#1}\vrule}
\hrule}}%end \toxit (\vtop used)
%
\def\bcell{\hbox to\csize\bgroup\vrule
height.8\csize depth.2\csize}
\let\ecell=}%TB 385, \egroup
%
\def\prc#1{\if*#1\cc\else\ifx\relax#1%
\cc\else%no crossed out cell
\ifnum`#1=\lccode`#1\low{#1}\else%
\cap{#1}\fi\fi\fi}%end \prc
\def\cc{\leaders\hrule height.8\csize
depth.2\csize\hfill}%end Crossed Cell
\def\low#1{\ifpuzzle\null\else\hfil%
\uppercase{#1}\fi\hfil}%end \low
\def\cap#1{\num\ifpuzzle\null%
\else\hfil#1\fi\hfil}%end \cap
%
\def\num{\global\advance\cnt1\relax%
\vbox to.8\csize{\rlap{\kern1pt%
\fiverm\the\cnt\hss}\vfil}}%end \num
%
%furthermore, in order to omit & and \cr
%in the data (natural input)
\def\markup#1{\ea\gdef\ea#1\ea{\ea%
\bfifol#1\efifol}}%end \markup
%
{\catcode`\^^M=13 %local scope
%Pick up and processing of lines
\gdef\bfifol#1^^M#2\efifol{%
\process{#1}%process line
\ifx\empty#2\empty\let\auxl=\lr%
\else\def\auxl{\rs\bfifol#2\efifol}%
\fi\auxl}        }%end local scope
%
\def\process#1{\bfifo#1\efifo}
%
%Pick up etc of chars per line
\def\bfifo#1#2\efifo{#1%put #1 back
\ifx\empty#2\empty\let\aux=\relax%
\else\def\aux{\cs\bfifo#2\efifo}%
\fi\aux}%end \bfifo...\efifo
%
\def\cs{&}\def\rs{\cr\noalign{\hrule}}
\def\lr{\cr}
%
%next is necessary to allow spaces
%for * (except in last column)
\def\bdata{\bgroup\obeylines%
\obeyspaces\store}
\def\store#1\edata{\egroup\def\data
{#1}}
{\obeyspaces\global\let =\relax}
\endinput    %27/3/92 cgl@rug.nl
```

[5] When they are provided along with the data, the invoke `\markup\data` is superfluous.
[6] Via nested use of `\bfifo...\efifo`, see FIFO and LIFO incognito

\crw

The crossword is implemented as a ruled \halign, with the framing added separately via \toxit. \offinterlineskip suppresses the space between the rows of the table. The explicit number of cells is not necessary to specify. The counter for the reference numbers is reset at the end.

\bcell...\ecell

The cell size is prescribed via the size of \hbox and \vrule. For the first column \vrule is set to width0pt, for the others the default width is used.

\prc

The macro typesets the cell contents according to the \data. (\halign takes care of the (inserted) &-s, \cr-s, and \noalign-s.) \if*#1 tests whether a crossed out cell has to be typeset, and if so \cc is invoked. For the other situation according to the case of the letter \low(er case letter) or \cap(ital letter), is invoked.

\cc

Crossed out cells are typeset as black cells via application of the leaders mechanism.[7]

\low, \cap

How the cell contents will be typeset depends upon the switch \ifpuzzle. The letters are typeset in upper case and centered. Note the invoke of \num.

\num

Generates and typesetes the reference numbers in the left upper corners.

\markup

The \data is processed per line via \bfifol...\efifol. The separator ^^M splits the data. The first line is delivered in the first argument and the rest in the second argument. This splitting up is repeated recursively.

\bfifol...\efifol

\bfifo...\efifo is invoked and \rs, respectively \lr, is added. \rs denotes row separator, and \lr stands for last row.

\bfifo...\efifo

Inserts column separators, \cs.

\bdata, \store

These store the user provided information between \bdata and \edata in \data, with the carriage return as active character.

3.2 Some pitfalls

In the macro \prc a test is needed to reveal the case of a letter. The PASCAL-like test \if\lowercase{#1}#1...[8] is a pitfall; \lowercase{\if#1}#1 and \ifnum`#1>96 work.

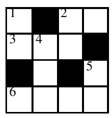
The functionality \bdata ⟨ *data* ⟩ \edata stores the user supplied information with active carriage returns. However,

```
\def\bdata{\bgroup\obeylines\gdef\data
        \bgroup}
\def\edata{\egroup\egroup}
```

is a pitfall (the \bgroup after \gdef must be an *explicit* brace!). In the examples the switch settings and the redefinition of \num and \rs, have been done *within* the math display, in order to keep the modifications local.

4 It is all in the game

Puzzle, clues and solution can be typeset all at once.



Across	Down
2 Switch mode	1 Public domain
3 Knuth	2 All right
6 Prior to TeX	4 All comes to it
	5 Atari type

is obtained—after \input crw.tex, data definition and markup—via

```
$$\vcenter{\noindent\sevenrm\csize=3ex
\puzzletrue\crw\data\parindent=3ex
\vtop{\hsize=21ex\obeylines Across
2 Switch mode
3 Knuth
6 Prior to \TeX}
\vtop{\hsize=25ex\obeylines Down
1 Public domain
2 All right
4 All comes to it
5 Atari type}
\fiverm\csize=3ex
\puzzlefalse\def\num{}\crw\data}$$
```

The automatic placement of crosswords in the text, as 'floating bodies,' is not dealt with. See therefore the general \figplace.tex, Zalmstra and Rogers (1989).

[7] This revealed a driver bug. \hbox to \csize {...\cc} in the crossword was not completely black, compare ■.
[8] Jürgen Knappen communicated the elegant solution \ifnum`#1 = \lccode`#1.... Victor Eijkhout suggested among others \ifnum \sfcode `#1 = 1000... (TeXbook p.76). The elegant solution is also in his book.

5 Variations

5.1 Black or ...

The appearance of the black cells can be changed via redefinition of `\cc`.

Solutions are also typeset just with letters at the appropriate places in the diagram; without rules and black cells.

P		O	N
D	E	K	
	N		S
E	D	I	T

This can be obtained after `\input crw.tex`, data definition and markup, via

```
\def\sol#1{\vtop{\offinterlineskip%
\halign{&\hbox to\csize{\vrule%
height.8\csize depth.2\csize%
width0pt\proc##}\cr%end preamble
#1}}}%end \sol
\def\proc#1{\if*#1\else\ifx\relax#1%
\else%no black cells
\hfil\uppercase{#1}\fi\fi\hfil}%end\proc
$$\def\rs{\cr}\sol\data$$
```

5.2 No solution diagrams

Why not supply the solution similar to the clues, as columns of text?

Across	Down
2 ON	1 PD
3 DEK	2 OK
6 EDIT	4 END
	5 ST

5.3 *-symbol

It is a minor adaptation to use another symbol to denote crossed out cells.

6 Conclusion

Is this just for FUN? I hope that the approach facilitates the use of TEX for typesetting crosswords. Moreover, the worked out ideas for inputting data WYSIWYG-like and have TEX to modify them appropriately for further processing, can be applied to typesetting of other games, for example bridge and chess, and in general to typesetting of (simple) tables with natural input.

It was easier to write the macro from scratch than to understand the LATEX macros of Hamilton Kelley. To my experience this is symptomatic for LATEX, and its derivatives.

From the application point of view the following general issues have been dealt with

- automatic mark up
- natural (and minimal) input
- automatic and hidden numbering
- ruled tables.

TEXnically use has been made of

- filling it up (`\leaders`, `\hrule`)
- testing case of letter (``\ifnum`#1 = \lccode`#1``)
- *local* catcode changes for making carriage return active (``\catcode`\^^M= 13 ``)[9]
- 2-part macros and storing the data provided in between by the user
- abstraction from `&` and `\cr`
- parameterizing via a switch and redefs
- testing for empty argument (`\ifx \empty #2 \empty...`)
- nested if-s
- replicator mechanism in preamble
- parameter separation
- FIFO recursion.

The author claims that crosswords can be typeset easily and of high-quality via TEX.

All the nitty-gritties put in reminded me of DEK[10]

> 'Yet dozen small refinements add up to something that is important to me, and I think such refinements might prove important to other people as well.'

7 Acknowledgements

I like to thank Jürgen Knappen for providing the elegant test for the case of a letter. David Salomon is kindly acknowledged for the stimulating (e-mail) discussions.

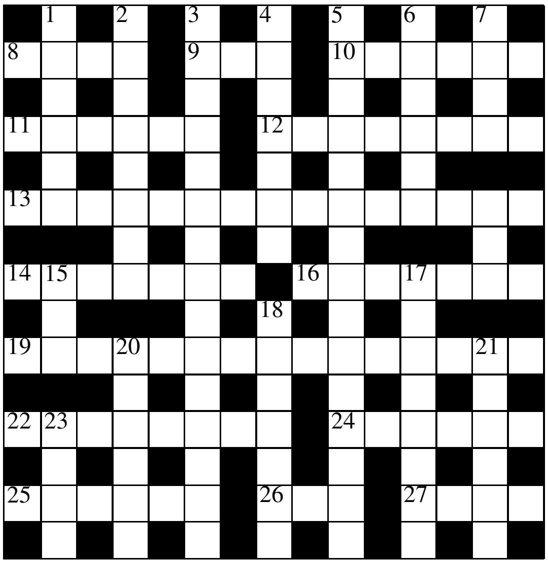
References

[1] Hamilton Kelley, B (1990): Some macros to draw crosswords. *TUGboat* 11, no. (1), 103 – 119.
[2] Knuth, D.E (1986): The TEXbook. Addison-Wesley.
[3] Laan, C.G. van der (submitted for publication): FIFO and LIFO incognito.
[4] Salomon, D (priv.comm.).
[5] Zalmstra, J, D.F. Rogers (1989): A page make-up macro. *TUGboat* 10, no. (1), 73 – 81.

[9] Also touched upon is to allow spaces for *'s, except at the line ends. This entailed making spaces active.
[10] Mathematical Typography, Bulletin of the AMS, 1, 2, 1979, p.345.

Appendix

Hamilton Kelley's puzzle[11]



is obtained—after `\input crw.tex`—via

```
\bdata%BHK's example
 S I C T D S P*
Swam Oho Icecap
 o p m r t n l*
Bopeep Schedule
 s l a i y u  *
Thalassographer
   e s n a   r*
HAirpin UmbRage
 r   o S b i  *
ScaLenetriangLe
   o a u c g e*
AMounted Allege
 a v e e l e a*
Floral Nil Tace
 l e y t y s y*
\edata\markup\data
$$\puzzletrue\crw\data$$
```

The above shows that the macros also take spaces for crossed out cells, except for those in the last column. It seems to be impossible to get hold of spaces at line ends in TEX, because TEX deletes endspaces, to accommodate for some computer series which fill up lines with trailing spaces. I could circumvent the S-state, where multiple spaces are skipped, TEXbook p.46.

The above is the next best to complete natural input for crosswords. If the spaces interfere with other parts of the document, the last line `{\obeyspaces...}`, of `crw.tex` can be deleted, and no spaces are accounted for in the crossword.

[11] Without BHK's clues, see for those his article.

LADIES and LATEX– II

Theo A. Jurriens

Kapteyn Astronomical Institute
P.O. Box 800
9700 AV Groningen
The Netherlands

taj@rugr86.rug.nl

Abstract

In dit artikel wordt mijn ervaring als LATEX-docent, vraagbaak voor secretaresses beschreven. Er zijn twee bronnen van problemen: organisatie en TEX-techniek.

1 Inleiding

Sinds een aantal jaren ben ik LATEX vraagbaak voor het secretariaat van de Afdeling Sterrenkunde van de Rijksuniversiteit Groningen. Ook de astronomen, veelal geen dame's, behoren tot mijn 'klantenkring'. In deze bijdrage wil ik mij richten op het gebruik van LATEX door secretaresses. Het is niet alleen gebaseerd op de ervaringen in Groningen en met de dames aldaar maar ook ervaring opgedaan tijdens diverse cursussen LATEX in Nederland en ver daar buiten (zie Appendix voor overzicht). Het artikel [1] geeft een overzicht van LATEX gebruik in Groningen.

De problemen inzake introductie en gebruik LATEX zijn twee-ledig: enerzijds zijn ze van organisatorische aard, anderzijds van technische en ook LATEX/TEXnische aard. Beide worden in deze bijdrage belicht.

2 Organisatie

In deze bijdrage beschouwen we de opdrachtgever van de secretaresse [1] als de klant, de secretaresse als de winkelier. Het LATEX/TEX-gedrag van de klant dwingt de winkelier tot het leren van LATEX: de klant krijgt ook steeds meer LATEX klanten: de grote uitgevers zien TEX en zeker LATEX tot **de standaard** voor de komende jaren. In de meeste gevallen leert de winkelier LATEX: de praktijk leert immers dat 95% van de klanten dan ook wel tevreden zijn. Maar ja de 5% ontevreden klanten kunnen het leven van de winkelier **zuur** maken. De winkelier moet altijd klant vriendelijk zijn, immers klant is koning. Daarnaast is de winkel altijd open en moet er uit 'voorraad' geleverd worden. Dat betekent in de praktijk dat de secretaresse nauwelijks tijd krijgt om LATEX te leren, laat staan om er rustig mee te oefenen.

Mijn cursussen geef ik dan ook zelden op de werkplek, wat uiteraard zijn nadelen heeft. Tenzij ik de garantie krijg van de 'klant' dat hij de winkelier enige dagen met rust laat. Genoemde cursus duurt minimaal twee dagen. Daarnaast waarschuw ik de klant dat hij of zij de winkelier leert om met haar nieuwe produkt om te gaan. Auteurs kunnen niet verwachten van een beginnend LATEX typiste dat zij het gelijk voor 100% beheerst. Dat doet de klant ja ook niet, zeker in het begin kan het voorkomen dat de klant meer van het produkt weet dan de winkelier. Ook houd ik altijd een pleidooi dat de winkelier beter beloond wordt. Goed TEX of LATEX vereist meer inzicht en vaardigheden van de winkelier dan zijn WP-collega.

Als de winkelier moderniseert van zijn WP-winkel naar zijn LATEX-winkel heeft dan enkele nadelen. Een LATEX-starter is veel beter, laatst genoemde is niet gehandicapt door WP-kennis. De bediening van de WP-winkel is veel vriendelijker voor de winkelier: immers de magische `F3` toets geeft alle informatie over de winkel. En veel winkeliers zijn opgeleid tot WP-handelaar. Zover mij bekend zijn er op erkende secretaresse opleidingen nog geen cursussen geavanceerd document opmaak. Kortom in veel gevallen moet de winkelier zijn TEX of LATEX adhoc leren. Maar gelukkig is er nog een vertegenwoordiger die af en toe de winkelier kan trainen. Maar ja als er problemen zijn, is hij niet altijd binnen handbereik. En de klant is er toch niet om de winkelier te helpen?

'Gouden regels' voor LATEX introductie op secretariaatsniveau:

1. geef de ruimte om LATEX te leren,
2. geef de ruimte om LATEX te oefenen, dus geen morgen af klussen in het begin,
3. iedere secretaresse een LATEX boek,

[1] de auteur is er zich van bewust dat ook mannen uiterst bekwaam als secretaris fungeren, dit artikel is ook zeker niet vrouw-onvriendelijk bedoeld.

4. een printer, om de mooie resultaten af te drukken, binnen handbereik,
5. overleg met de secretaresse voor het inleveren van het te typen manuscript,
6. vertrouw op de LATEX ervaring van de secretaresses en wees een plezierige 'klant',
7. gebruik LATEX ook voor het simpele werk: het niet-formule werk. Op deze manier wordt de 'winkelier' gedwongen één standaard te hanteren,
8. zorg voor de juiste beloning.

3 TEX-techniek

Eerder is genoemd dat WP een gebruikersvriendelijke tekstverwerker is. Voor secretaresses, vers van de opleiding is het de standaard. Voor een efficiënt TEX of LATEX gebruik is een intelligente, gebruikersvriendelijke editor gewenst: een menu gestuurde emacs, een emacs á la Beebe of een speciale optie in WP: 'saven as TeX'. Overigens de nieuwe versie van K-Talk (de WP naar TEX conversie) schijnt goed[2] te werken. Sommige secretaresse's gebruiken WP als editor, werkt goed maar het gevaar bestaat dat de files per ongeluk niet als ASCII bewaard worden. In *TUGboat* is eerder bericht over WP als intelligente LATEX-editor. Een ander 'gevaar' is dat als WP aanwezig is dat het gebruikt wordt voor het simpele werk.

Op een gegeven moment ontstaat de behoefte om zelf definities of eigen styles files te maken: pure TEX kennis is dan nodig maar niet altijd aanwezig. Bestaande style files zijn onleesbaar en bovendien is het LATEX boek voor dit doel onbegrijpelijk. Het is een veel gehoorde klacht: in het boek van Lamport kun je niets vinden. Het wordt versterkt door onvoldoende kennis van de Engelse TEX-termen.

Het maken van serie-brieven (mailing) is voor een secretaresse niet voor de handliggend. Een lokale TEX-specialist is nodig. Evenzo geldt dit voor het uptodate houden van de software. Een secretaresse kan veelal niet verantwoordelijk worden gesteld voor systeemonderhoud.

De eerder genoemde 'Gouden regels' kunnen we als volgt aanvullen:
1. zorg voor goede apparatuur en onderhoud van software,
2. een goede editor,
3. een lokale TEX of LATEX vraagbaak.

Naast deze regels, tips is er nog een taak voor de diverse TEX User Groups. Maak een internationale dictionary van TEX, LATEX termen. Een goed boek, in de eigen taal, is er nog niet voor deze doelgroep[3]. Adresbeheer à la BIBTEX, serie brieven. Een problemen rubriek van en voor deze doelgroep in de nationale TEX organen! In de volgende MAPS de eerste aflevering: *problemen met dames.*

References

[1] Theo Jurriens, LADIES *and LATEX*, MAPS 91.2, 102–104.
[2] Theo Jurriens, *TEXniques in Siberia*, MAPS 91.2, 87–90.

LATEX-cursus ervaring:

- Secretariaat Sterrenkunde, Lab. voor Ruimteonderzoek Groningen,
- Wiskunde en Informatica, Rijksuniversiteit Groningen,
- Vakgroep Econometrie, Rijksuniversiteit Groningen,
- Kernfysisch Versneller Instituut, KVI Groningen,
- Vakgroep Econometrie, Katholieke Universiteit Brabant,
- Vakgroep Natuurkunde, Universiteit van Amsterdam,
- Rekencentrum en Universiteit Novosibirsk.

[2] niet zelf getest
[3] een reeds lange gekoesterde wens van me.