Od przyjaciół

Syntactic Sugar

Kees van der Laan*

Abstract

A plea is made for being honest with TEX and not imposing alien structures upon it, otherwise than via compatible extensions, or via (non-TEX) user interfaces to suit the publisher, the author, or the typist. This will facilitate the process to get (complex) publications out effectively, and typographically of high-quality.
Keywords (nested) loop, switch, array addressing, plain TEX, macro writing, education.

Introduction

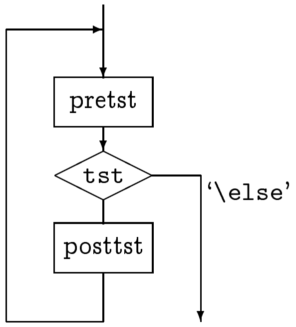
TEX is a formatter and also a programming language. TEX is different from current high-level programming languages, but very powerful. A class on its own, and therefore unusual, and unfamiliar.
Because of TEX being different, macro writers propose to harnass TEX into a more familiar system, by imposing syntaxes borrowed from various successful high-level programming languages. In doing so, injustice to TEX's nature might result, and users might become intimidated, because of the difficult—at least unusual—encoding used to achieve the aim. The more so when functional equivalents are already there, although perhaps hidden, and not tagged by familiar names. This is demonstrated with examples about the loop, the switch, array addressing, optional and keyword parameters.
Furthermore, TEX encodings are sometimes peculiar, different from the familiar algorithms, possibly because macro writers are captivated by the mouth processing capabilities of TEX. Users who don't care so much about TEX's programming power, but who are attracted by the typesetting quality, which can be obtained with TEX as formatter, can be led astray when in search for a particular functionality they stumble upon unusual encodings. They might conclude that TEX is too difficult, too error-prone and more things like that and flee towards *Wordwhatever*, or embrace Desk Top Publishing systems.

* This is an abridged version of the paper submitted to EuroTEX'93

The way out is education, next to the provision of compatible, well-documented and supported user interfaces, which don't act like syntactic sugar, by neglecting, or hiding, the already available functional equivalents. Neither the publication of encodings, nor the provision of encodings via file servers or archives—although a nice supporting feature for the TEX-ies—is enough. The quality, compatibility and the simplicity of the (generic) macros should be warranted too.
It is not the aim of this paper to revitalize a programming languages notation war, but to stimulate awareness, and exchange ideas.

Loops

DEK's loop, TEXbook p.219, implements the flow



with syntax

`\loop⟨pretst⟩\iftst⟨posttst⟩\repeat.`

Special cases result when ⟨pretst⟩, or ⟨posttst⟩ is empty. The former is equivalent to for example PASCAL's **while** ...**do** ..., and the latter to **repeat**...**until**. With this awareness, I consider the variants as proposed by for example Pittman, [22], and Spivak, [26], as syntactic sugar.
If `\ifcase...` is used, then we have for ⟨posttst⟩ several parallel paths, of which one—determined dynamically—will be traversed. Provide and choose your path! What do you mean by traversing the `\else`-path?

Why another loop?

Kabelschacht, [10], and Spivak, [26], favour a loop which allows the use of `\else`.[1] I have some objections to Kabelschacht's claim that his loop is a *generalization* of plain's loop.
First, it is not a generalization, just a clever, but variant, implementation of the loop flow chart.

[1] Their loops are equivalent to the general form of the loop with the execution of an extra part after the loop.

Second, it is not compatible with plain's loop. His exit path is via the `\then` branch (or via any of the `\or`-s, when `\ifcase` is used), and not via the `\else` branch.

The reason, I can think of, for introducing another loop, while the most general form has been implemented already, is the existence of commands like `\ifvoid`, and `\ifeof`, and the absence of their negatives `\ifnvoid`, respectively `\ifneof`. In those cases we like to continue the loop via the `\else` branch. For the latter case this means to continue the loop when the file is *not* ended. This can be attained via modifying the loop, of course, but I consider it simpler to use a `\newif` parameter, better known as 'boolean' or 'logical' in other programming languages. With the `\newif` parameter, `\ifneof`, the loop test for an end of file—functionally ¬`\ifeof`—can be obtained via

```
\ifeof\neoffalse\else\neoftrue\fi\ifneof
```

For an example of use, see the Sort It Out subsection. Related to the above encoding of the logical ¬, are the encodings of the logical and, ∧, and or, ∨, via

Functional code	TEX encoding
¬`\if...`	`\if...\notfalse\else`
	`\nottrue\fi\ifnot`
`\if...`∧`\if...`	`\andtrue\if...\if...`
	`\else\andfalse`
	`\else\andfalse\fi\fi`
	`\ifand`
`\if...`∨`\if...`	`\ortrue`
	`\if...\else\if...\else`
	`\orfalse\fi\fi   \ifor`

with the `\newif`-s: `\ifnot`, `\ifand`, and `\ifor`.

Nesting of loops

Pittman, [22], argued that there is a need for other loop encodings.

> 'Recently, I encountered an application that required a set of nested loops and local-only assignments and definitions.
> TEX's `\loop...\repeat` construction proved to be inadequate because of the requirement that the inner loop be grouped.'

If we take his (multiplication) table—I like to classify these as deterministic tables, because the data as such are not typed in—to be representative, then below a variant encoding is given, which does not need Pittman's double looping. The table is typographically a trifle, but it is all about how the deterministic data are encoded. My approach is to consider it primarily as a table, which it is after all. Within the table the rows and columns are generated, via recursion, and not via the `\loop`. Furthermore, I prefer to treat rules, a frame, a header and row stubs as separate items to be added to the table proper, [15]. The *creation* of local quantities is a general TEX aspect. I too like the idea of a hidden counter, and the next best TEX solution via the local counter. The local versus global creation of counters is a matter of taste, although very convenient now and then. The creation of local quantities is tacitly discouraged by DEK's implementation, because there is no explicit garbage collector implemented and therefore no memory savings can be gained. The only thing that remains is protection against programming mistakes, which is indeed important.

Pittman's table, focused at the essential issue of generating the elements, can be obtained via

```
$$\vbox{\halign{&\ \hfil#\hfil\strut\cr
\rows}}$$
```

with

```
\newcount\rcnt\newcount\ccnt\newcount\tnum
\newcount\mrow\newcount\mcol \mrow2 \mcol3
\def\rows{\global\advance\rcnt1
  \global\ccnt0\cols\ifnum\rcnt=\mrow\swor
  \fi\rs\rows}
\def\swor#1\rows{\fi\crcr}
\def\cols{\global\advance\ccnt1
  \tnum\rcnt\multiply\tnum\ccnt\the\tnum
  \ifnum\ccnt=\mcol\sloc\fi\cs\cols}
\def\sloc#1\cols{\fi}
\def\rs{\cr}\def\cs{&}
```

The result is

1	2	3
2	4	6

The termination of the recursion is unusual. It is similar to the mechanism used on page 379 of the TEXbook, in the macro `\deleterightmost`. The latter TEXnique is elaborated in [4], and [16].

The above shows how to generate in TEX deterministic tables, where the table entries in other programming languages are generally generated via nested loops. One can apply this to other deterministic math tables—trigonometric tables for example—but then we need more advanced arithmetic facilities in TEX (or inputting the data calculated by other tools), not to mention the appropriate mapping of tables which extend the page boundaries.

For a more complete encoding see Table Diversions, [15]. The idea is that rules and a frame be commanded via `\ruled`, and `\framed`. The header via an appropriate definition of `\header`, ×, the indication that we deal with a multiplication table, in `\first`, and the row stubs via definition of the row stub list. All independent and separate from the table proper part.

A better example of a nested loop is for example the encoding of bubble sort as given in [17].

Loops and novices

Novice TEXies find DEK's loop unusual, so they sugar it into the more familiar **while**, **repeat**, or **for** constructs, encouraged to do so by exercises as part of courseware. From the functionality viewpoint, there is no need for another loop notation.

With respect to the **for** loop, I personally like the idea of a hidden counter, [13], and [22]. The hidden counter has been used in an *additional* way to plain's loop in for example [13] (via `\preloop` and `\postloop`), and will not be repeated here. This way of doing is a matter of taste, which does not harm, nor hinder, because it is a compatible extension.

And, ...for the nesting of loops we need scope braces, because of the parameter separator `\repeat`. If braces are omitted, the first `\repeat` is mistaken for the outer one, with the result that the text of the outer loop will *not* become the first `\body`. The good way is, to make the inner `\repeat` invisible at the first loop level, by enclosing the inner loop in braces.

With non-explicit nesting—for example the inner loop is the replacement text of a macro—we still need scope braces, because otherwise the `\body` of the outer loop will be silently redefined by the body of the inner loop.

The point I like to get accross is, that there is no real need for another loop encoding. Syntactic sugar? Yes!

Switches, is there a need?

Apart from the `\ifcase...` construct, TEX seems to lack a multiple branching facility with symbolic names. Fine, [4], introduced therefore

```
\def\fruit#1{\switch\if#1\is
a \apple
b \banana
c \cherry
d \date      \end}
```

I have 2, or rather 3, remarks to the above.
First, the 'switch'-functionality is already there.
Second, Fine's implementation is based upon

> 'It is clear that `\switch` must go through the alternatives one after another, reproducing the test...'

Well, ...going through the alternatives one after another is not necessary.
Third, his example, borrowed from Schwarz, [24], can be solved more elegantly without using a 'switch' or nested `\if-s` at all, as shown below.
The first two aspects are related. Fine's functionality can be obtained via

```
\def\fruit#1{\csname fruit#1\endcsname}
%with
\def\fruita{\apple}
\def\fruitb{\banana} %et cetera
```

With for example:
`\def\apple{{\bf apple}},`
`\fruit a`
yields
apple.

And what about the 'else' part? Thanks to `\csname`, `\relax` will return when the control sequence has not yet been defined. So, if nothing has to happen we are fine. In the other situations one could define `\def\fruitelse{...}`, and make the else fruits refer to it, for example `\def\fruity{\fruitelse}`, `\def\fruitz{\fruitelse}`, etc. When the set is really uncountable we are in trouble, but I don't know of such situations. And, ...the five letters 'fruit' are there only to enhance uniqueness of the names.

As example J. Fine gives the problem, treated by Schwarz, [24], to print vowels in bold face.[2]

The problem can be split into two parts. First, the general part of going character by character through a string, and second, to decide whether the character at hand is a vowel or not.

For the first part use for example, `\dolist`, TEXbook Exercise 11.5, or `\fifo`, [16].

```
\def\fifo#1{\ifx\ofif#1\ofif\fi#1\fifo}
\def\ofif#1\fifo{\fi}
```

[2] A bit misplaced example because the actions in the branches don't differ, except for the non-vowel part.

For the second part, combine the vowels into a string, aeiou, and the problem is reduced to the question $\langle char \rangle \in$ aeiou? Earlier, I used the latter approach when searching for a card in a bridge hand, [12].[3] That was well-hidden under several piles of cards, I presume? Recently, I have used the same method for recognizing accents and control sequences in a word, [17]. Anyway, searching for a letter in a string can be based upon \atest, TEXbook, p.375, or one might benefit from \ismember, p.379. I composed the following

```
\newif\iffound
\def\loc#1#2{%locate #1 in #2
\def\locate##1#1##2\end{\ifx\empty##2%
\empty\foundfalse\else\foundtrue\fi}
\locate#2.#1\end}
```

Then

```
\fifo Audacious\ofif
```

yields

Aud**a**c**iou**s, with

```
\def\process#1{\uppercase{\loc#1}%
{AEIOU}\iffound{\bf#1}\else#1\fi}
\def\fifo#1{\ifx\ofif#1\ofif\fi
  \process#1\fifo}
```

Note that en-passant we also accounted for uppercase vowels. By the way, did you figure out why a period—a free symbol—was inserted between the arguments for \locate? It is not needed in this example.[4] Due to the period one can test for substrings: $string_1 \in string_2$? Because, $\{string_1 \in string_2\} \wedge \{string_2 \in string_1\} \Rightarrow \{string_1 = string_2\}$, we also have the possibility to test for equality of strings, via \loc. Happily, there exists the following straightforward, and TEX-specific, way of testing for equality of strings

```
\def\eq#1#2{\def\st{#1}\def\nd{#2}
\ifx\st\nd\eqtrue\else\eqfalse\fi}
```

For lexicographic comparison, see [17], [16].

Knuth's switches

Don Knuth needed switches is his manmac macros—\syntaxswitch,\xrefswitch etc.—TEXbook,p.424. He has implemented the functionality via nested \if-s. My approach can be used there too, but with some care with respect to { token in \next (read: some catcode adaptations). For example[5]

```
\ea\def\csname sw[\endcsname{[-branch}
\ea\def\csname sw|\endcsname{bar-branche}
%etc. then
\def\next{[}\csname sw\next\endcsname, and
\def\next{|}\csname sw\next\endcsname
```

yields: **[-branch**, and **bar-branche**.

For manmac see the TEXbook, p. 412–425, and the discussion [19].

Array addressing

Related to the switch, or the old computed goto as it was called in FORTRAN, is array addressing. In TEX this can be done via the use of \csname. An array element, for example elements identified among others in PASCAL by a[1] or a[apple], can be denoted in TEX via the control sequences

```
\csname a1\endcsname
%respectively
\csname aapple\endcsname
```

For practical purposes this accessing, or should we say 'reading,' has to be augmented with macros for writing, as given in [5], and [7]. Writing to an array element can be done via

```
\def\a#1#2{\ea\def\csname a#1%
\endcsname{#2}}\a{1}{Contents}
```

yields Contents, after the above.

The point I like to make is, that 'array addressing'—also called table lookup by some authors—is already there, although unusual and a bit hidden, but, ... we are used to things like strong type-checking, isn't? Once we can do array addressing we can encode all kind of algorithms, which make use of the array data structure. What about sorting? See the Sort It Out subsection, for a glimpse, and the in depth treatment, [17], with $O(n \log n)$ algorithms, and application to glossary and index sorting.

Conclusion

It is hoped that authors who can't resist the challenge to impose syntaxes from successful programming languages upon TEX, also encode the desired functionality in TEX's peculiar way, and contrast this with their proposed improvements. The novice, the layman and his peers will benefit from it.

The difficulties caused by TEX's unusual encoding

[3] The macro there was called \strip.

[4] If omitted the find of 'bb' in 'ab' goes wrong: abbb vs. ab.bb, will be searched.

[5] \ea – an abbreviation for \expandafter

mechanisms, can best be solved via education, and not via imposing structures from other languages. The latter will entail confusion, because of all those varieties. Furthermore, it is opposed to the Reduced Instruction Set idea, which I like. For me it is similar to the axioms-and-theorems structure in math, with a minimal number of axions, all mutual orthogonal.

Publishing houses, user groups, and macro writers are encouraged to develop and maintain 'user interfaces,' which do justice to TEX's nature, and don't increase the complexity of TEX's components. Good examples are: TUGboat's sty files, AMS-LATEX & AMS-TEX, and LAMS-TEX, and not to forget good old manmac! Macro-TEX and lxiii[6] are promising.

File servers and archives are welcomed, but the compatibility, the simplicity and in general the quality, must be warranted too. Not to mention pleasant documentation and up-to-date-ness.

My wishful thinking is to have intelligent local archives, which have in store what is locally generally needed, and know about what is available elsewhere. The delivery should be transparent, and independent whether it comes from elsewhere or was in store.

References

[1] Beebe, N.H.F (1991): The TUGlib server. MAPS 91.2, 117–123.

[2] Beeton, B.N, R. Whitney (1989): *TUGboat* 10, no. (3), 378–385.

[3] Eijkhout, V (1991): TEX by Topic. A-W.

[4] Fine, J (1992): Some basic control macros for TEX. *TUGboat* 13, no. (1), 75–83.

[5] Greene, A. M (1989): TEXreation—Playing games with TEX's mind. TUG89. *TUGboat* 10, no. (4), 691–705.

[6] Hendrickson, A (1989): MacroTEX.

[7] Hendrickson, A (1990): Getting TEXnical: Insights into TEX macro writing techniques. *TUGboat* 11, no. (3), 359–370.

[8] Jeffreys, A (1990): Lists in TEX's mouth. *TUGboat* 11, no. (2), 237–244.

[9] Jensen, K, N. Wirth (1975): PASCAL user manual and report. Springer-Verlag. *TUGboat* 11, no. (2), 237–244.

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

[11] Knuth, D.E (1984): *The TEXbook*, A-W.

[12] Laan, C.G van der (1990): Typesetting Bridge via TEX. *TUGboat* 11, no. (2), 265–276.

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

[14] Laan, C.G van der (1992b): FIFO & LIFO incognito. EuroTEX '92, 225–234. Also MAPS92.1. An elaborated version is FIFO & LIFO sing the BLUes.

[15] Laan, C.G van der (1992c): Table Diversions. EuroTEX '92, 191–211. A little adapted in MAPS92.2.

[16] Laan, C.G van der (1992d): FIFO & LIFO sing the BLUes. MAPS92.2, 139–144. (To appear TUGboat, 14, 1.)

[17] Laan, C.G van der (1993a): Sorting in BLUe. MAPS93.1. (21p. Heap sort encoding is released in MAPS92.2.)

[18] Laan, C.G van der (1993b): Typesetting number sequences. MAPS93.1. (4p.)

[19] Laan, C.G van der (1993c): Manmac BLUes. MAPS93.1. (In progress in the same series: AMS BLUes, and TUGboat BLUes.)

[20] Lamport, L (1986): LATEX, user's guide & reference manual. A-W.

[21] Maus, S (1991): An expansion power lemma. *TUGboat* 12, no. (2), 277.

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

[23] Salomon, D (1992): NTG's Advanced TEX course: Insights & Hindsights. MAPS 92 Special. 254p.

[24] Schwarz, N (1987): *Einführung in TEX*, A-W.

[25] Siebenmann, L (1992): Elementary Text Processing and Parsing in TEX. *TUGboat* 13, no. (1), 62–73.

[26] Spivak, M.D (1987): LAMS-TEX. TEXplorators.

[27] Youngen, R.E (1992): TEX-based production at AMS. MAPS92.2. 7p.

Kees van der Laan jest prezesem prężnej, holenderskiej grupy użytkowników TEX-a (NTG), z którą GUST utrzymuje ścisły kontakt.

[6] The LATEX3 project.