Syntactic Sugar

Kees van der Laan
Hunzeweg 57, 9893PB, Garnwerd, The Netherlands, 05941-1525
Internet: cgl@risc1.rug.nl

Abstract

A plea is made for being honest with TeX and not imposing alien structures upon it, other than via compatible extensions, or via (non-TeX) user interfaces to suit the publisher, the author, or the typist. This will facilitate the process of producing (complex) publications effectively, and typographically of high-quality.

Introduction

TeX is a formatter and also a programming language. TeX is unlike traditional high-level programming languages. It is still powerful, in a class of its own, unusual, and unfamiliar.

Because TeX is different, macro writers propose harnessing it 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, and mouth versus stomach processing.

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, while searching 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 Word*whatever*, or embrace Desk Top Publishing systems.

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 TeXies — 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.

First, I'll glance at the big collections, and after that I'll dive into the details of macros from various sources.

In the Large

Let me first look roughly at the big collections, and refer for more details to papers on the issue.

In my opinion the math mark-up in Spivak's *AMS*-TeX is syntactic sugar. It claims to be essentially simpler than plain's math mark-up, which it is not. It is just different and does not provide more facilities than plain. A proof? All the examples provided in "The Joy of TeX" can be formatted equally within plain. In LA*MS*-TeX the table part and the commutative diagrams are substantial extensions of plain, next to the general symbolic reference scheme. For more details see my book review of Spivak's Œuvre.

I consider amsppt.sty and the like an adaptation by a publisher of manmac for production, with value added, if not for the user's guides and the provided support. These latter things can't be overestimated for Ben Lee User[1] in my opinion. For more detail see my *AMS BLUes*.

Furthermore, I consider LATeX as syntactic sugar, especially the math part. LATeX 2.x is even more dangerous because it claims to be perfect, which it is not. If I compare the mark-up in the spirit of *The TeXbook* with my mark-up obtained via LATeX, then the latter is much more verbose and has not added much. The extras like the picture environment, symbolic and cross-referencing, and the bibliography

---
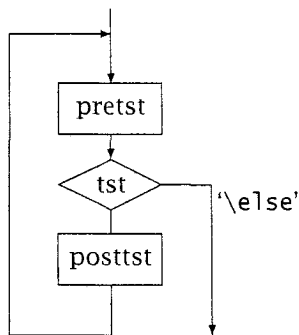
[1] From *The TeXbook* fame, I like the nickname BLU.

environment, can be added easily by independent tools in a manmac-like basis, when needed.[2] Multi-column issues have in general their difficulties — more likely buses-and-weirdness effects will occur; see Richard Southall's contribution about these issues at this conference — but if one is willing to adapt proofs by hand now and then, it can be added because the functionality is available as a separate tool, nowadays.

> If only manmac and Knuth's other example formats had been appropriately documented in (additional) user's guides, then the (LA)TEX world would have looked much different from what it is today.

In the Small

In the sequel I'll descend into detail and discuss: loops, switches, array addressing, optional and keyword parameters, mouth processing, sorting and lexicographic comparison.

Loops. Knuth's loop, (*The TEXbook*, p. 219), implements the general flow

with (pseudo) syntax

$\loop\langle pretst\rangle\if\ldots\langle posttst\rangle\repeat.$

Special cases result when either $\langle pretst\rangle$ or $\langle posttst\rangle$ 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 (1988) and Spivak (1991) as syntactic sugar.

If \ifcase... is used, then we have for $\langle posttst\rangle$ 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?

---

[2] With respect to the mark-up of the list of references it is such a waste that every author should supply the full mark-up. Why not just supply references to the database of pre-formatted entries, in possession of and maintained by the editors?

Why another loop? Kabelschacht (1987) and Spivak (1989, 1991) favour a loop which allows the use of \else.[3] 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. 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 \ifnonvoid and \ifnoneof, respectively. 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,[4] \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 (1988) 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.'

---

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

[4] Be aware that the implementation of \newif does not allow for \global.

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, (van der Laan, 1992c). 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 Knuth'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 p. 379 of *The TEXbook*, in the macro \deleterightmost. The latter TEXnique is elaborated in Fine (1992) and van der Laan (1992d).

The above shows how to generate in TEX deterministic tables, where the table entries in other programming languages are generally generated via nes-

ted 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 my *Table Diversions* (van der Laan, 1992c). The idea is that rules and a frame be commanded via \ruled, respectively \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 van der Laan (1993a).

Loops and novices. Novice TEXies find Knuth'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, see van der Laan (1992a) or Pittman (1988). The hidden counter has been used in an *additional* way to plain's loop in, for example, van der Laan (1992a), (via \preloop and \postloop), and will not be repeated here. This method 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 would like to get across 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 (1992) introduced therefore

```
\def\fruit#1{\switch\if#1\is
a \apple
```

```
b \banana
c \cherry
d \date        \end}
```

I have two, or rather three, 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...'

Going through the alternatives one after another is not necessary. Third, his example, borrowed from Schwarz (1987), can be solved more elegantly without using a 'switch' or nested \ifs 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 Fine gives the problem, treated by Schwarz (1987), of printing vowels in bold face.[5]

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, (*The TEXbook*, ex 11.5), or \fifo, (van der Laan, 1992d).

```
\def\fifo#1{\ifx\ofif#1\ofif\fi
    \process{#1}\fifo}
\def\ofif#1\fifo{\fi}
% with to be defined by the user
\def\process#1{...}
```

For the second part, combine the vowels into a string, aeiou, and the problem is reduced to the

---

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

question ⟨*char*⟩ ∈ aeiou? Earlier, I used the latter approach when searching for a card in a bridge hand (van der Laan, 1990).[6] 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, (van der Laan, 1993a). Anyway, searching for a letter in a string can be based upon \atest, (*The TEXbook*, p. 375), or one might benefit from \ismember, on p. 379. I composed the following

```
\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}        \newif\iffound
```

Then \fifo Audacious\ofif yields **Audacious**, with

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

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.[7] Due to the period one can test for substrings: $string_1 \in string_2$? Because, $\{string_1 \in string_2\} \wedge \{string_2 \in string_1\} \Rightarrow \{string_1 = string_2\}$, it is also possibile 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 van der Laan (1992d, 1993a) or Raichle (1992).

Knuth's switches. Knuth needed switches in his manmac macros — \syntaxswitch, \xrefswitch and the like — (*The TEXbook*, p. 424). He has implemented the functionality via nested \ifs. My approach can be used there too, but with some care with respect to the {-token in \next (read: some catcode adaptations). For example:

```
\ea\def\csname sw[\endcsname{[-branch}
\ea\def\csname sw|\endcsname{bar-branche}
\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 in van der Laan (1993c).

---

[6] The macro there was called \strip.

[7] If omitted the search for 'bb' in 'ab' goes wrong: abbb vs. ab.bb, will be searched.

Kees van der Laan

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
\csname aapple\endcsname
```

For practical purposes this accessing, or should we say 'reading', has to be augmented with macros for writing, as given in Greene (1989) and Hendrickson (1990). Writing to an array element can be done via

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

Typesetting (reading) via \csname a1\endcsname yields Contents, after the above.

The point I would like to make is that 'array addressing' — also called table look-up by some authors — is already there, although unusual and a bit hidden. However, we are used to things like strong type-checking, aren't we? 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 in van der Laan (1993a), with $O(n \log n)$ algorithms, and application to glossary and index sorting.

Keyword parameters. In TeX literature the functionality of keyword parameters is heavily used. Some authors impose the syntax known from command languages upon TeX: for examples see Appelt (1987) or Siebenmann (1992). In my opinion this is syntactic sugar, because of the following rhetorical question. What is essentially the difference between

```
\ref
\key W\by A. Weil
\paper Sur ...
...
\endref
```

as detailed in Siebenmann (1992) and, for example,

```
{\def\key{W}\def\by{A. Weil}
\def\paper{Sur ...}...
\typeset}
```

The typesetting is done in the cited case by \ref...\endref, and in the alternative case by \typeset. The values for the keys are the background defaults and those temporarily redefined. Note that in both cases the order of the specifications is free and that defaults (empty) are used, for not explicitly specified values.

In my bordered table macro (van der Laan, 1992c), I could have introduced keyword parameters obeying the command languages syntax. Happily, I refrained from that. I needed several parameters. A parameter for framing, with functionalities nonframed, framed, and dotframed. A parameter for ruling, with functionalities nonruled, ruled, hruled, vruled, and dotruled. And a parameter for positioning of the elements, with functionalities centered, flushed left, and flushed right. (The first element of each enumerated list of values, acting as the default value.)

Furthermore, I decided to provide the user the possibility of *optionally* specifying a caption, a header, a rowstub list, or a footer. If any of these is not explicitly specified, then the item will be absent in print too.[8] This resembles optional parameter behaviour, but has been realized by Knuth's parameter mechanism.

In following Knuth's approach, I succeeded in keeping the encoding compact, and transparent. I find it as simple, direct, and serving the purpose extremely well.[9]

Optional parameters. Among others, in LaTeX, (Lamport, 1986), the mechanism of optional parameters is used. Optional parameters are a special case of keyword parameters. Knuth used optional/keyword parameters abundantly, and called them just parameters, as opposed to arguments of macros. (Think for example of his various parameters and his \every...s.) So it is already there, although in an unusual way.

Another example which illustrates the arbitrariness of the syntax choice with respect to optional/keyword parameters vs. Knuth's parameters is TUGboat's \twocol vs. LaTeX's twocolumn style option.

Intriguing optional parameter conventions are the general and the systematic encoding of $\mathcal{A}_{\mathcal{M}}S$-TeX's \nofrills, and TUGboat.sty's \@checkoptions.[10]

Salomon's plain Makeindex. At NTG's '92 spring meeting David Salomon reported about adapting MakeIndex to work with plain. He used optional

---

[8] Another difficulty was to provide a default template, which can be overridden by the user. This was solved by the same approach.

[9] Earlier, I had a similar experience (van der Laan, 1990).

[10] More about these issues in AMS BLUes (van der Laan, 1993d) and TUGboat BLUes, (van der Laan, 1993e) respectively.

parameters, with the function as given in the following table

Source	Typeset	
	document	index
^[abc]	abc	abc
^[xyz]{abc}	abc	xyzabc
^|abc|	abc	abc
^|\abc|	\abc	\abc
^{\abc}	replacement text of \abc	same
^[|\abc|!xyz]{}	nothing	\abc, xyz

and combinations thereof.

The same functionality can be obtained via Knuth's parameter mechanism. Only one parameter is needed. Let us call this the token variable \p. The idea is that the contents of \p have to be inserted before the index-entry in the index, and *not* in the text. Some symbols can be given a special meaning, like Salomon did, for example with ! (to denote a subentry).

Salomon's Source	Alternative
^[abc]	^{abc}
^[xyz]{abc}	{{\p{xyz}^{abc}}}
^|\abc|	^{|\abc|}
^{\abc}	^{\abc}
^[|\abc|!xyz]{}	{\p{|\abc|!xyz}^{}}

In the above | denotes manmac's verbatim delimiter. The macro for ^ has to be adapted accordingly. It is beyond the scope of this paper to work that out in detail.[11] The point I like to make is that the specification can be done equally well, if not simpler, via Knuth's parameter mechanism. In manmac, Knuth provides simple mark-up facilities for writing index reminders to a file, except for comments and see..., and see also... parts. The latter can be accounted for. I have touched upon these issues in *Manmac BLUes* (van der Laan, 1993c).

Mouth vs. stomach. When one starts with macro writing in TEX one can't get around awareness of TEX's digestive processing. Mouth processing is unusual. For the moment, I consider it as a special kind of built-in pre-processing, an unusual but

---

[11] The preparation of an index via TEX has gotten a new dimension since my encodings of sorting within TEX. Also the writing of general index reminders to a file has been elaborated upon. For the first, see van der Laan (1993a), and for the latter, see van der Laan (1993c).

powerful *generalization* of the elimination of 'dead branches.'[12]

Now and then encoding is published in *TUGboat*, and other sources as well, which looks difficult, and which does not seem to reflect the familiar algorithms. Sometimes, it has become difficult, because of the sought-after processing in the mouth, see for example, Jeffrey (1990) and Maus (1991).[13] The latter author agrees more or less with what is stated above '...although the macros are hard to read...'.

> What puzzles me are the following questions.
>
> Why don't authors provide the straightforward TEX encoding, not restricted to mouth processing, as well?
>
> Why don't they make clear the *need* for mouth processing, or should I say mouth optimization?
>
> If so, why don't they start with the straightforward encoding and explain the adaptation steps?

Faced with the above questions myself, I would answer that it is apparently too difficult to do so.[14] Furthermore, I read and worked on the math parts, the alignment parts, the macro chapter, a substantial part of the dirty tricks Appendix D and of the example formats Appendix E of *The TEXbook*, and until now found only a comment about the *capability* of TEX's mouth processing along with the macro \deleterightmost. I know of the argument that there is a need for it within an \edef, a \write..., and the like. I have heard that, but from an application point of view, my obvious answer is: Isn't it

---

[12] Knuth might forgive me my ignorance at this point. My brows are raised when I see published code, restricted to mouth processing, which looks so verbose and unintelligible. I definitely turn my back on it when the straightforward alternative encoding is familiar, compact, elegant and generic, despite the rumour that TEX's mouth has the programming power of the Turing machine. As it is, in my opinion, that is something different from, let us say, literate programming, to indicate a broad stream of readable programs.

[13] By the way, when do we know that something is completely processed in the mouth? Is there a check on it? Or, is it just an abstract part of the TEXnigma?

[14] And what about the efficiencies? From the viewpoint of the machine and with respect to human understanding? I have not seen the common and mouth versions of an algorithm published simultaneously, let alone have them compared with respect to timing.

Kees van der Laan

possible to do the things outside those constructs, equally well, and pass through the results?

> If authors don't help me out with the above, I consider the encoding as *l'art pour l'art.* Nothing wrong with that, on the contrary. The only thing against that is that it will spread a negative image about TeX encoding, certainly not under the theoretical computer scientists, but under the day-to-day BLUe-type programmers, if not the authors who just use (LA)TeX to get their work out, beautifully.

Agreed, Maus referred to *The TeXbook,* but Jeffrey could have provided a more intelligible solution, and should have refrained from burying his method under a sort of program correctness math. At the moment, it is easier to start from scratch. I experienced that already with the encoding of: the Tower of Hanoi, typesetting crosswords, generating n-copies, lexicographic comparison, and sorting. The published encodings inspired me to develop alternatives, sure, but that should not be the aim, should it? Furthermore, I wonder how many *users* have been discouraged by those 'difficult to read' codes, especially when the familiar codes are straightforward?

n-copies. I needed Maus' functionality — avant la lettre — in typesetting a fill-in form, where a number of rows had to be repeated. Of course, my editor can do it — statically — and that served the purpose. It is easy for sure, but it does not look elegant. A straightforward use of tail recursion satisfied me better, because of the simplicity, the compactness and the elegance, at the expense of a negligible efficiency loss. See the example about the bridge form in *Table Diversions* (van der Laan, 1992c).[15] The tail recursion determines the number of copies dynamically, as do the other solutions given by Knuth, for example the nice solution via the use of \aftergroup, (*The TeXbook,* p. 374).[16]

Sort it out. Jeffrey's problem is: given an unsorted list of (positive) integers via symbolic names, typeset the ordered list.[17] In order to concentrate on the main issues, assume that his list adheres to Knuth's

list structure (*The TeXbook,* p. 378). As example consider the list[18]

```
\def\lst{\\\ia\\\ib\\\ic}
\def\ia{314}\def\ib{27}\def\ic{1}
```

The sorted numbers **1, 27, 314,** are obtained via

```
\def\\#1{\ifnum#1<\min\let\min=#1\fi}
\def\first#1{\def\lop\\##1##2\pol{%
        \let\min=##1}\ea\lop#1\pol}
\newif\ifnoe
\loop\ifx\empty\lst\noefalse\else
     \noetrue\fi
\ifnoe  \first\lst   \lst   \min,
  {\def\\##1{\ifx##1\min\else\noexpand\\%
     \noexpand##1\fi}\xdef\lst{\lst}}%
\repeat
```

The encoding implements the looping of the basic steps

- find minimum (via \lst, and suitable definition of the active list separator \\)
- typeset minimum (via \min)
- delete minimum from the list (again via an(other) appropriate definition of the active list separator).

For removing a typesetted element, I was inspired by \remequivalent (*The TeXbook,* p. 380).[19]

The above is effective for short lists, as was the case in Jeffrey's application.[20] For longer (and random) lists, techniques of order $O(n \log n)$ are more appropriate. For plain TeX encodings for the latter see van der Laan (1993a). There it has been applied to lexicographic sorting, too.

Lexicographic comparison. Eijkhout has provided macros — focused at mouth processing — for lexicographic ordering (1991). His \ifallchars ...\are ...\before made ample use of \expandafter, and is not easily accessible for somebody with say two years of TeX experience.[21]

Hackers might go into ecstasy, but application-oriented users become discouraged. For a straightforward alternative, not restricted to mouth processing, see van der Laan (1992d). The point I'd like

---

[15] The complexity is of order $O(n)$, instead of $O(\log n)$, which is not important, because of the small number of copies involved.

[16] Knuth in his chart macro — for typesetting font tables — uses also the straightforward approach of supplying all the lines in \normalchart. He could have used recursion similar to the way I did it in the multiplication table of Pittman.

[17] I have also worked on this problem, taking care of the range notation aspects (van der Laan, 1993b).

[18] Equally-well, the comma could have been used as an active list separator, which looks more natural. I decided to adhere to Knuth's notation.

[19] I was not able to apply the parameter separator technique to locate the element to be removed.

[20] Remember that sorting based on linear search has complexity $O(n^2)$.

[21] Moreover it had a flaw, as pointed out by Bernd Raichle (1992), who presented an alternative with less \expandafters and an intriguing use of \csname.

to make is that I would have welcomed the familiar solution and the transformation steps as well.

Conclusions

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 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 axioms, all mutually orthogonal.

Publishing houses, user groups, and macro writers are encouraged to develop and maintain user interfaces[22] which do justice to TeX's nature, and don't increase the complexity of TeX's components. Good examples are: TUGboat's sty files, $\mathcal{A}_{\mathcal{M}}S$-LaTeX, $\mathcal{A}_{\mathcal{M}}S$-TeX, and LaTeX-TeX, and not to forget good old manmac! Macro-TeX and the LaTeX3 project 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 archives,[23] which have in store what is locally generally needed, and know about what is available elsewhere. The delivery should be transparent, and independent of whether it comes from elsewhere or was in store. For corrections and certifications I would welcome a similar approach as ACM's loose-leaf collection of algorithms

> Any third-rank engineer can make a complicated apparatus more complicated, but it takes a touch of genius to find one's way back to the basic principles, which are normally fairly simple.
> E.F. Schumacher, *Small is beautiful.*

I'm happy to include the following synopsis of the TUG '93 proceedings referee

> 'The point he is trying to make is that TeX macros are software and the really difficult

---

[22] And user's guides.

[23] Essentially the trickle approach, see the Earn Network Resource Guide (1993), from the fileserver.

lessons of software engineering should be used by TeX macro writers as well. Those of us who try in software engineering are not overly successful in keeping things simple and it is not surprising that little of the right way of doing software has been included in the construction of TeX macros.'

Acknowledgements

Bibliography

Appelt, W. "Macros with Keyword Parameters". *TUGboat*, 8(2), 182 – 184, 1987.

Appelt, W. "TeX für Fortgeschrittene — Programmiertechniken und Makropakete". Addison-Wesley, 1988.

Beebe, N.H.F. "The TUGlib Server". *MAPS*91.2, 117 – 123, 1991. (Also in TeXline 11.)

Beeton, B.N, R.F Whitney. "TUGboat Author's Guide". *TUGboat*, 10(3), 378 – 385, 1989. (Not completely up-to-date.)

EARN Association "Guide to Network Resource Tools", 1993.

Eijkhout, V. "TeX by Topic". Addison-Wesley, 1992.

Fine, J. "Some basic control macros for TeX". *TUGboat*13(1), 75 – 83, 1992.

Greene, A.M. "TeXreation—Playing games with TeX's mind". *TUGboat*, 10(4), 691 – 705, 1989.

Hendrickson, A. "MacroTeX". 1989.

Hendrickson, A. "Getting TeXnical: Insights into TeX Macro Writing Techniques". *TUGboat*, 11(3), 359 – 370, 1990. (Also *MAPS*90.2.)

Jeffrey, A. "Lists in TeX's mouth". *TUGboat*, 11(2), 237 – 244, 1990.

Jensen, K., and N. Wirth. "PASCAL user manual and report". Springer-Verlag, 1975.

Kabelschacht, A. "\expandafter vs. \let and \def in Conditionals and a Generalization of plain's \loop". *TUGboat*, 8(2), 184 – 185, 1987.

Knuth, D.E. "*The TeXbook*". Addison-Wesley, 1984.

Laan, C.G van der. "Typesetting Bridge via TeX". *TUGboat*, 11(2), 265 – 276, 1990.

Laan, C.G van der. "Math into BLUes". Part I: Mourning. Proceedings TUG '91, *TUGboat*, 12(4), 485 – 501, 1991.
Part II: Sing Your Song. Proceedings EuroTeX '91, *GUTenberg Cahiers*, 10&11, 147 – 170, 1991.

Kees van der Laan

Laan, C.G van der. "Tower of Hanoi, Revisited". *TUGboat*, **13**(1), 91 - 94, 1992.

Laan, C.G van der. "FIFO & LIFO Incognito". EuroTeX '92, 225 - 234, 1992. (Also in *MAPS*92.1. An elaborated version is FIFO & LIFO Sing the BLUes.)

Laan, C.G van der. "Table Diversions". EuroTeX '92, 191 - 211, 1992. (A little adapted in *MAPS*92.2.)

Laan, C.G van der. "FIFO & LIFO Sing the BLUes". *TUGboat*, **14**(1), 54 - 60, 1993. (Also in *MAPS*92.2, 139 - 144, 1992.)

Laan, C.G van der. "Spivak's Œuvre". *MAPS*92.1, 139 - 142, 1992.

Laan, C.G van der. "Sorting in BLUe". These Proceedings. (The complete article is in *MAPS*93.1, 149 - 170. Heap sort encoding has been released in *MAPS*92.2.)

Laan, C.G van der. "Typesetting number sequences". *MAPS*93.1, 145 - 148, 1993.

Laan, C.G van der. "Manmac BLUes — Or how to typeset a book via TeX". *MAPS*93.1, 171 - 191, 1993. (To be submitted TUG '94)

Laan, C.G van der. "AMS BLUes — professionals at work". *MAPS*93.1, 192 - 212, 1993.

Laan, C.G van der. "TUGboat BLUes — how TeXies do it". *MAPS*93.2 (In progress).

Lamport, L. "LaTeX, user's guide & reference manual". Addison-Wesley, 1986.

Maus, S. "An expansion power lemma". *TUGboat*, **12**(2), 277, 1991.

Pittman, J.E. "Loopy.TeX". *TUGboat*9(3), 289 - 291, 1988.

Raichle, B. In: V. Eijkhout. "Oral TeX: Erratum". *TUGboat*, **13**(1), p. 75, 1992.

Salomon, D. "NTG's Advanced TeX course: Insights & Hindsights". *MAPS*92 Special, 1992. Revised ≈ 500p.

Schwarz, N. "Einführung in TeX". Addison-Wesley, 1987.

Siebenmann, L. "Elementary Text Processing and Parsing in TeX—The Appreciation of Tokens". *TUGboat*, **13**(1), 62 - 73, 1992.

Spivak, M.D. "$\mathcal{A}_\mathcal{M}S$-TeX — The Joy of TeX". American Mathematical Society, 1986.

Spivak, M.D. "L$^A\mathcal{M}S$-TeX—The Synthesis". TeXplorators, 1989.

Spivak, M.D. "L$^A\mathcal{M}S$-TeX Wizard's manual". TeXplorators, 1991.

Youngen, R.E. "TeX-based production at AMS". *MAPS*92.2, 63 - 68, 1992.