BIJLAGE X

The structure of the TEX processor [1] [2]

Victor Eijkhout

Center for Supercomputing Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
eijkhout@csrd.uiuc.edu

Februari, 1991

The inner workings of TEX are explained by its author [1] in terms of an analogy with the digestive tract. Apart from the fact that this gives rise to a whole genre of jokes[3], the analogy becomes definitely strained when regurgitation takes place in the mouth, or when the eyes take part in the process.

In this article I will describe the TEX processor as a multi-layered engine that successively transforms characters into tokens, tokens into lists, and from these lists builds a typeset page.

1 Four TEX processors

The way TEX processes its input can be viewed as happening on four levels. One might say that the TEX processor is split into four separate units, each accepting the output of the previous stage, and delivering the input for the next stage. The input of the first stage is then the `tex` input file; the output of the last stage is a `dvi` file.

For many purposes it is most convenient and insightful to consider these four levels of processing as happening after one another, each one accepting the *completed* output of the previous level. In reality this is not true: TEX is not something like a four-pass compiler. All levels are simultaneously active, and there is interaction between them.

The four levels are

1. The input processor. This is the piece of TEX that accepts input lines from the file system of whatever computer TEX runs on, and turns them into tokens. These are mostly character tokens that comprise the typeset text, and control sequence tokens that are commands to be processed by the next two levels.
2. The expansion processor. A number of tokens generated in the first level – macros, conditionals, and a number of primitive TEX commands – is subject to expansion. Expansion is the process that replaces some (sequences of) tokens by another (possibly empty) sequence.
3. The execution processor. Control sequences that are not expandable are executable, and this execution takes place on the third level of the TEX processor. One part of the activity here concerns changes to TEX's internal state: assignments and macro definitions are typical activities in this category. The other thing going on on this level is the construction of horizontal, vertical, and mathematical lists.
4. The visual processor. In the final level of processing the visual part of TEX processing is performed. Here horizontal lists are broken into paragraphs, vertical lists are broken into pages, and formulas are built out of math lists. Also the output to the `dvi` file takes place on this level. The algorithms working here are not accessible to the user, but they can be influenced by a number of parameters.

2 The input processor

The input processor is that part of TEX that translates whatever characters it gets from the input file into tokens. The output of this processor is a stream of tokens: a token list. Most tokens fall into two categories: character tokens and control sequence tokens. The remaining category is that of the parameter tokens; these will not be treated here.

[1] To be published in TUGboat, © 1991, TEX Users Group.
[2] This is a chapter from my book 'TEX by Topic', to be published by Addison-Wesley.
[3] Tokens being 'sicked up again' [2], output being 'TEXcrement' [3], or the particularly deplorable title of [4] . . .

2.1 Character input

For simple input text, characters are made into character tokens. However, TEX can ignore input characters: a row of spaces in the input is usually equivalent to just one space. Also, TEX itself can insert tokens that do not correspond to any character in the input, for instance the space token at the end of an input line, or the `\par` token after an empty line.

Not all character tokens represent characters that are to be typeset. Characters fall into sixteen categories – each one specifying a certain function that a character can have – of which only two contain the characters that will be typeset. The other categories contain such characters as {, }, `&`, and `#`. A character token can be considered as a pair of numbers: the character code – usually the ASCII code – and the category code.

When the escape character \ appears in the input, TEX's behaviour in forming tokens is more complicated. Basically, TEX builds a control sequence by taking a number of characters from the input and lumping them together into a single token.

The behaviour with which TEX's input processor reacts to category codes can be described as finite-state automaton with three internal states: N, new line, M, middle of line, and S, skipping spaces. These states and the transitions between them are treated in chapter 8 of *The TEXbook*.

2.2 Two-level input processing

TEX's input processor is in fact even a two-level processor. Due to limitations of the terminal, the editor, or the operating system, the user may not be able to input any desired character. Therefore, TEX provides a mechanism to access with two superscript characters all of the available character positions. This may be considered to be a separate stage of TEX processing, taking place prior to the three-state finite automaton mentioned above.

For instance, the sequence `^^+` is replaced by `k` because the ASCII codes of `k` and `+` differ by 64. Since this replacement takes place before tokens are formed, one may write `\vs^^+ip 5cm` to get the effect of `\vskip 5cm`. More useful examples than this exist.

Note that this first stage is a transformation from characters to characters, without considering category codes. These come into play only in the second phase of input processing, where characters are converted to character tokens by coupling the category code to the character code.

3 The expansion processor

TEX's expansion processor accepts a stream of tokens and, if possible, expands the tokens in this stream one by one until only unexpandable tokens remain. Macro expansion is the clearest example of this: if a control sequence is a macro name, it is replaced (together possibly with parameter tokens) by the definition text of the macro.

Input for the expansion processor is provided mainly by the input processor. The stream of tokens coming from the first stage of TEX processing is subject to the expansion process, and the result is a stream of unexpandable tokens which is fed to the execution processor.

However, the expansion processor comes into play also when an `\edef` or `\write` is processed. The parameter token list of these commands is expanded as if the lists would have been on top level, instead of the argument to a command.

There is a special fascination to macros that work completely by the expansion processor. See the recent articles [4], [5], and [6] for some good examples.

3.1 The process of expansion

Expanding a token comprises the following steps:

- See if the token is expandable.
- If the token is unexpandable, pass it to the token list currently being built, and take on the next token.
- If the token is expandable, replace it by its expansion. For macros without parameters, and a few primitive commands such as `\jobname`, this is indeed a simple replacement. Usually, however, TEX needs to absorb some argument tokens from the stream in order to be able to form the replacement of the current token. For instance, if the token was a macro with parameters, sufficiently many tokens need to be absorbed to form the arguments corresponding to these parameters.
- Go on expanding, starting with the first token of the expansion.

Deciding whether a token is expandable is usually a simple decision. Macros and active characters, conditionals, and a number of primitive TEX commands (see the list on page 215 of *The TEXbook*) are expandable, other tokens are not. Thus the expansion processor replaces macros by their expansion, it evaluates conditionals and eliminates any irrelevant parts of these, but tokens such as `\vskip` and character tokens, including characters such as dollars and braces, are passed untouched.

3.2 Special cases: `\expandafter`, `\noexpand`, and `\the`

As stated above, after a token has been expanded TEX will start expanding the resulting tokens. At first sight the `\expandafter` command would seem to be an exception to this rule, because it expands only one step. What actually happens is that the sequence

`\expandafter`<token$_1$><token$_2$>

is replaced by

<token$_1$><*expansion of token*$_2$>

and this replacement is in fact reexamined by the expansion processor.

Real exceptions do exist, however. If the current token is the `\noexpand` command, the next token is considered for the moment to be unexpandable: it is handled as if it were `\relax` (more about this control sequence follows below), and it is passed to the token list being built.

Example: in the macro definition

```
\edef\a{\noexpand\b}
```

the replacement text `\noexpand\b` is expanded at definition time. The expansion of `\noexpand` is the next token, with a temporary meaning of `\relax`. Thus, when the expansion processor tackles the next token, the `\b`, it will consider that to be unexpandable, and just pass it to the token list being built, which is the replacement text of the macro.

Another exception is that the tokens resulting from `\the`<token variable> are not expanded further if this statement occurs inside an `\edef` macro definition.

3.3 Braces in the expansion processor

Above, it was said that braces are passed as unexpandable character tokens. In general this is true. For instance, the `\romannumeral` command is handled by the expansion processor; when confronted with

```
\romannumeral1\number\count2 3{4
```

TEX will expand until the brace is encountered: if `\count2` has the value of zero, the result will be the roman numeral representation of `103`.

As another example,

```
\iftrue {\else }\fi
```

is handled by the expansion processor as if it were

```
\iftrue a\else b\fi
```

The result is a character token, be this a brace or a letter.

However, in the context of macro expansion the expansion processor will recognize braces. First of all, a balanced pair of braces marks off a group of tokens to be passed as one argument. If a macro has an argument

```
\def\macro#1{ ... }
```

one can call it with a single token

```
\macro 1 \macro \$
```

or with a group

```
\macro {abc} \macro {d{ef}g}
```

Secondly, when the arguments for a macro with parameters are read, no expressions with unbalanced braces are accepted. In

```
\def\a#1\stop{ ... }
\a bc{d\stop}e\stop
```

the argument is `bc{d\stop}e`. Only balanced expressions are accepted here.

4 The execution processor

The execution processor builds lists: horizontal, vertical, and math lists. Corresponding to these lists, it works in horizontal, vertical, or math mode. Of these three modes 'internal' and 'external' variants exist. In addition to building lists, this part of the TEX processor also performs mode-independent processing, such as assignments.

Coming out of the expansion processor is a stream of unexpandable tokens to be processed by the execution processor. From the point of view of the execution processor, this stream contains two types of tokens:

- Tokens that signal an assignment (this includes macro definitions), and other tokens that are independent of the mode, such as `\show` and `\aftergroup`.
- Tokens that build lists: characters, boxes, and glue.

Some objects can be used in any mode, for instance boxes can appear in horizontal, vertical, and math lists. The effect of such an object will of course still depend on the mode. Other objects are specific for one mode. For instance, characters (to be more precise: character tokens of categories 11 and 12) are intimately connected to horizontal mode: if the execution processor is in vertical mode when it encounters a character, it will switch to horizontal mode.

For the expansion processor a character token is just an unexpandable object. On this level, however, something is actually done with it. Some characters are typeset, but the execution processor can also encounter, for instance, math shift characters (usually `$`), or braces. When a math shift character is found in the stream of tokens, math mode is entered (or exited if the current mode was math mode); when a left brace is found, a new level of grouping is entered.

One control sequence handled by the execution processor deserves special mention: `\relax`. This control sequence is not expandable, but the execution is 'empty'. Compare the effect of `\relax` in

```
\count0=1\relax 2
```

with that of `\null` defined by

```
\def\null{}
```

in

```
\count0=1\null 2
```

In the first case the expansion process that is forming the number stops at `\relax` because it is unexpandable, and the number `1` is assigned. In the second case `\null` expands to nothing, so `12` is assigned.

5 The visual processor

TEX's visual processor encompasses those algorithms that are outside direct user control: paragraph breaking, alignment, page breaking, math typesetting, and `dvi` file generation. Various parameters control the operation of these parts of TEX.

Some of these algorithms return their results in a form that can be handled by the execution processor. For instance, a paragraph that has been broken into lines is added to the main vertical list as a sequence of horizontal boxes with intermediate glue and penalties. Also, the page breaking algorithm stores its result in `\box255`, so output routines can disect it. On the other hand, a math formula can not be broken into pieces, and, of course, shipping a box to the `dvi` file is irreversible.

6 Further examples

6.1 Skipped spaces

Skipped spaces provide an illustration of the view that TEX's levels of processing accept the completed input of the previous level. Consider the commands

```
\def\a{\penalty200}
\a 0
```

Faulty reasoning

"The `\a` is encountered, expanded, the space then delimits the number"

would lead to the conclusion that this is equivalent to `\penalty200 0`. It is not. Instead, what results is

```
\penalty2000
```

because the space after `\a` is skipped in the input processor.

6.2 Internal quantities and their representations

TEX uses various sorts of internal quantities, such as integers and dimensions. These internal quantities have an external representation, which is a string of characters, such as `4711` or `91.44cm`.

Conversions between the internal value and the external represenation take place on two different levels, depending on the direction the conversion goes. A string of characters is converted to an internal value in assignments such as

```
\pageno=12 \baselineskip=13pt
```

or statements like

```
\vskip 5.71pt
```

and all of these statements are handled by the execution processor.

On the other hand, the conversion of the internal values into a representation as a string of characters is handled by the expansion processor. For instance,

```
\number\pageno \romannumeral\year
\the\baselineskip
```

are all processed by expansion.

Note that in the `\baselineskip` example above the conversion from string of characters to internal value was 'automatic'. The conversion the other way has to be forced by a command such as `\number`. Thus there is no danger that the sequence

```
\pageno=3 \count\MyCount=\pageno 5
```

will result in assigning either 15 or 35 to `\MyCount`.

As a final example, suppose `\count2=45`, and consider the statement

```
\count0=1\number\count2 3
```

The expansion processor tackles `\number\count2` to give the characters `45`, and the space after the `2` is absorbed because it only serves as a delimiter of the number of the `\count` register. In the next stage of processing, the execution processor will then see the statement

```
\count0=1453
```

and execute this.

References

[1] Donald Knuth, *The TEXbook*,. Addison–Wesley Publishing Company, 1984.

[2] Angela Barden, Some TEX manuals, TUGboat 12(1991), no. 1.

[3] Ron Whitney, private communication.

[4] Victor Eijkhout, Oral TEX, TUGboat 12(1991), no. 2.

[5] Alan Jeffrey, Lists in TEX's mouth, TUGboat 11, no. 2, 237–245.

[6] Sonja Maus, Looking ahead for a ⟨box⟩, TUGboat 11, no. 4, 613–614.

BIJLAGE Y

Comments on the Future of TEX and METAFONT [1]

Nelson H.F. Beebe
Center for Scientific Computing
Department of Mathematics
220 South Physics Building
University of Utah
Salt Lake City, UT 84112
USA
Tel: (801) 581-5254
FAX: (801) 581-4148
Beebe@science.utah.edu

October 1990

1 Introduction

Donald E. Knuth's article, "The Future of TEX and METAFONT", elsewhere in this issue[2], clearly states the Grand Wizard's wishes about these programs and the Computer Modern font family.

Where does that leave TUG? The opening paragraph of TUG's bylaws includes this statement (the emphasis is mine):

> ...specifically to identify, develop, operate, fund, support, promote and encourage charitable, educational and scientific programs and projects which will stimulate those who have an interest in *systems for typesetting technical text and font design*; to exchange information of same and associated use of computer peripheral equipment; to establish channels to facilitate the exchange of macro packages, etc., through publications and otherwise; and to develop, implement and sponsor educational programs, seminars and conferences in connection with the foregoing...

I believe that this expressly says that TUG's purview legitimately goes beyond TEX, METAFONT, and Computer Modern, whose further development has been frozen by their author in the interests of providing a constant solid base for their users, and of returning to his own extensive research and writing efforts, which have been outstanding landmarks in the development of the fields of Computer Science and Applied Mathematics.

2 TEX is international

As the TEX-related portion of the Utah bibliography project described in my President's message in this issue of *TUGboat* will attest, the use of TEX is widespread. Many books and journals are routinely typeset by TEX, including almost all of the publications of the American Mathematical Society, one of the world's largest publishers of mathematical material. Large on-line data bases in TEX input form now exist.

I suggest that no other typesetting system, or desk-top publishing system, has been used for as many languages as TEX has. TEX is in use for all major European languages, plus Arabic, Chinese, Coptic (Ethiopian), Hebrew, several Indian languages, Japanese, Persian, Russian, Thai, Turkish, Vietnamese, and likely others that I may be unaware of. This list includes languages that are written horizontally and vertically. TEX can support typesetting of multiple languages in the same text, thanks to the work of Frank Liang on hyphenation [11], of Michael Ferguson on multi-lingual TeX [4, 5, 6, 7], and of Donald Knuth and Pierre MacKay on TEX-XET [9].

These research efforts led to several features incorporated in TEX 3.0 to make multilingual typesetting standardly available. For related work in other typesetting systems, see [2] on tri-directional typesetting, and articles in the July 1987, August 1988, and May 1990 issues of the Communications of the ACM.

There are textbooks about TEX in at least Danish, Dutch, English, French, German, and Japanese, and I know of

[1] Reprinted from TUGboat 11 (1990), No. 4 – © 1990, TEX Users Group; reprinted with permission.
[2] Appendix GG of NTG MAPS 90.2.

in-progress translations to Persian of the TEXbook and the LATEX User's Guide and Reference Manual.

There are TUG members in nearly 50 countries, and I'm sure there are TEX users in many more. Besides TUG, there are five thriving regional groups in Western Europe, and five or more others are forming.

3 The challenge from desk-top publishing systems

The international use of TEX suggests that Donald Knuth's decision to freeze further development will in some ways be highly beneficial. However, it does *not* imply that TEX, METAFONT, and Computer Modern are the last word in computer-based typesetting. If TUG does not pursue further development of typesetting software, TEX may be doomed to extinction far sooner than it should, for several reasons:

- Desk-top publishing is *big* business, with several tens of millions of installed personal computers forming the potential market base. The Salt Lake Tribune on 10 October 1990 carried an article on Utahns included in the just-released Forbes list of the 400 wealthiest people in the world. The two developers of Word Perfect, one of the most popular word processing systems available on personal computers, workstations, and some mainframes, have a combined worth of nearly one (North American) billion dollars; the young chairman of Microsoft Corporation is worth even more.
- Software is a commodity that is relatively cheap to produce and distribute. The actual development costs of most commercial software are only a small fraction of potential sales revenues, and the computing industry has numerous examples of the quick attainment of fabulous wealth. What *does* cost a lot of money is sales and marketing, and the on-going support of software, including personnel, authoring, and documentation. This situation encourages competitiveness and rapid development of new products.
- Desk-top publishing (WYSIWYG)[3] systems are attractive to many people, particularly novices, because of the immediate feedback that they provide. With most of them, it is impossible to generate syntax errors of the type that TEX is perhaps infamous for, because input is checked character by character as it is entered, and formatting commands are generated by function keys and menu selections, rather than as embedded markup. Few of these systems today are suited to the batch typesetting required in journal and periodical production, because they bind a graphical input and output interface too tightly to the typesetting machinery; however, that market, because of its publishing volume, will eventually prove attractive.
- Users of most WYSIWYG systems are encouraged by the immediate feedback of the typeset display to make visual, rather than logical, design decisions. Design professionals often criticize visual design [10, Section 1.4] because it can lead to poor typography. Also, the visual layout may make it difficult to re-use the text, or to reformat it for a different output style. These objections may disappear as newer generations of these systems provide better support for document styles, and separation of the jobs of authoring or document entry, and document design.
- Several desk-top publishing systems are already capable of easily handling multi-column output, multi-column floats, flowing of typeset text around inserts (both rectangular and non-rectangular), and easy integration of graphics with text; these are areas where TEX is noticeably deficient.

4 TEX's advantages

In view of the points raised in the preceding section, we must then ask what does TEX (and I mean also METAFONT, Computer Modern, and related software) offer that competing desk-top publishing systems do not, at least not yet?

- TEX provides public-domain access to the source code of its related software. Source code of commercial implementations remains proprietary, but the changes from the public domain versions are usually in system-dependent areas that do not affect the overall operation of the software, and for most machines, both public domain and commercial implementations are available.

  Public access to the source code is extremely important. It permits both low-cost, or even free, public-domain implementations, and supported commercial implementations, of TEX to be available on many different platforms. A commercial user of TEX need not be tied to any single vendor of the software; such ties can become a significant competitive disadvantage when the supplier does not keep up with technological progress. As one such example, I cite the TV Guide experience [1].

  Although TEX is probably one of the most bug-free software packages of its size, it is reassuring to a user to know that if a question ever arises as to why the system typeset text in a particular way, the availability of well-documented source code makes it in principle possible to find the reason. Public access to source code means that bugs are often found and reported by several users, and fixes can come more quickly. By contrast, commercial desk-top publishing systems are almost always unfathomable black boxes whose surprises are indecipherable; it may be difficult to convince a vendor that an anomaly is a 'bug' instead of a 'feature'.
- TEX source code is written in a relatively portable language, and consequently, it is available *today* for vir-

[3] WYSIWYG = What You See Is What You Get, sometimes called What You See Is All You've Got.

tually every commercially-available computing system, from personal computers, up to supercomputers.

- The wide availability and use, and the frozen development, of TEX mean that we can view it as an *archival* document formatting system. Most commercial publishing products have completely ignored this issue; succeeding product generations offer new features and bug fixes, but are often incompatible with earlier ones. It is certainly true that much of what is published today is "throw-away" material, and in such cases, whether the publishing system can reformat the same document years from now is of no concern.
  However, in academic circles, this is decidedly not the case. Academicians research and write in the interest of wide dissemination of their ideas, both to current colleagues, and to future generations. Authors and publishers of such material are interested in re-using it for multiple documents. One of the TEX90 speakers from a major publisher noted that in some fields of study, the same text can be re-used more than a dozen times.
- TEX's freedom from architectural and commercial licensing restrictions facilitates collaborative efforts of several authors to work on the same document, even if they have different computer hardware.
- TEX's markup is visible, not hidden in magical undocumented binary data embedded in the document. This has several virtues:
  - Detection and correction of formatting errors is usually easier when the formatting commands can be seen.
  - It is relatively easy to write simple filters that strip the markup from a document to produce raw text which is input to other software tools for spell checking, grammatical analysis, and so on.
  - The markup is recorded in the same character set as the raw text, greatly facilitating document exchange between unlike systems, or via electronic mail.
- TEX's support for visible markup means that translation may be possible between it and other markup systems, such as SGML-based ones.
- TEX supports a powerful macro language that permits the creation of separate input interfaces that can be quite different from `plain` TEX. AMS-TEX and LATEX are the most obvious examples, but the Free Software Foundation's TEXinfo and LATEXinfo systems, and the use of TEX as the typesetting engine for documents written in other markup languages, as is done at at least two major publishing houses, are other examples. Most desk-top publishing systems lack this extensibility.
- TEX is capable of handling multi-lingual typesetting; few commercial publishing systems today can make this claim.
- TEX's mathematical typesetting abilities are still unmatched by most desk-top publishing systems. Its Computer Modern font family, together with the AMS font extensions, provides a repertoire of characters that is far more comprehensive than almost anything available on other systems. (I was able to announce at the Cork meeting that Adobe Systems has finally released a Lucida font in POSTSCRIPT format with a set of mathematics characters matching Computer Modern. Lucida is the font used in the typesetting of *Scientific American*.) The public-domain nature of TEX will of course make it possible for commercial systems to incorporate TEX's sophisticated algorithms for mathematics; however, this is likely to happen slowly because most of the commercial desk-top publishing market has little need for mathematical typesetting.
- TEX, and other systems based on visible markup (including those that use SGML), have a significant advantage over WYSIWYG systems in that style and content can be clearly separated. In most desk-top publishing systems, style and content are inextricably entwined. This has important ramifications for alternate uses of the input text, for user training, and for the effort needed to change the style without modifying the content.
  With TEX, authors and clerical staff need learn only *one* system that can be used with very minor changes to produce documents in a wide variety of styles.

5 Some observations

TEX currently has a portability advantage over most other typesetting systems. Many commercial publishing products are tied very closely to the hardware or window system architecture of a specific machine, particularly in the personal computer market. This has meant years of delay in getting them ported to other systems. The rise of the C language, particularly during the 1980s, as an efficient, but nevertheless portable, machine-independent implementation language is slowly beginning to be recognized by vendors. Assembly-language coded systems are now being rewritten in C or C++ to reach a wider market. Recent examples include SAS, Word Perfect, and Lotus 1-2-3. Because of the spread of popular window systems, such as X, Microsoft Windows, and others, and the efforts to standardize them, I expect that by the end of this decade, most commercial software products related to publishing will be available on as wide a range of machines as TEX currently is.

While it is true that standard TEX does not provide an immediate visual display of the typeset text, the Berkeley VORTEX project, about which too little has been written, and ArborText's Publisher system are demonstrations that TEX *can* have such an interface. The rapid advances in computer speeds that have occurred, largely through RISC processor developments, and the volume production economizations possible through sales of millions of personal computers, suggest that we are only a few short years away from instantaneous typeset on-line display.

Few existing systems, including WYSIWYG ones and

TEX, are suitable for newspaper publishing, which is characterized by its complicated layout of text and graphics in up to six or eight columns, and daily deadlines that cannot be missed without serious economic impact. I expect that the most printing done in the world today is in newspapers. While most of larger newspapers now use computer-based typesetting, I suspect that their systems are rather specialized for that industry.

6 Necessary future developments

The preceding sections have discussed the relative strengths and weaknesses of TEX versus desk-top publishing systems. I have found in discussions with other TUG members at meetings, and in mail exchanges, that many of us share the view that development of TEX cannot stand still. Donald Knuth has placed understandable restrictions on the use of the names TEX, METAFONT, and Computer Modern. Consequently, evolutionary systems arising from TEX will have to use different names.

I believe strongly that what needs to be done now is for those users of TEX and METAFONT who have pushed the limits of those systems to begin writing down detailed descriptions of just what those limitations are, and to make well thought-out suggestions about the directions that future work ought to take.

I made a start last year on the relation of TEX and graphics in [3].

Frank Mittelbach gave a wonderfully incisive exposition on the future of TEX at the College Station TUG'90 meeting [12], and followed that at the Cork TEX'90 conference with a fine presentation of work done together with Reinhard Wonneberger on the future of BIBTEX [14].

Michael Vulis has shown with an actual implementation [13] how scalable fonts tightly integrated into a TEX-like system can offer new and interesting capabilities. To those who would quibble with his incorporation of the name TEX, I would observe that VTEX is a superset of TEX, and with a special command-line argument, it will disable all extensions and perform exactly like TEX; nevertheless, it would be advisable to adhere to the Grand Wizard's wishes, and change the name.

John Hobby presented some very promising work at the Stanford TUG'89 meeting on extensions of METAFONT for generation of POSTSCRIPT output [8], and related work by Shimon Yanai and Daniel Berry should soon appear in *TUGboat*.

We need more such articles! Please, if you can contribute new ideas, and I know from personal contacts that many of you can, write them down (or even up) for publication in *TUGboat* or other journals in the field.

Only when we have a solid base of written contributions from the TEX experts will it be possible for some future researcher to have a reliable starting point for the design of the evolution of TEX to the next generation of typesetting system, and that person will have the added challenge of finding new names!

Let us hope that a major design goal of such an effort will be the maintenance of compatibility with existing TEX and METAFONT input, so that the substantial, and growing, base of existing TEX and METAFONT material will continue to be processable, with exactly the same results, by the next generation of computer-based typesetting systems. I believe that this would be far preferable to having separate, but mutually incompatible, systems that must try to coexist peacefully.

Incompatibility may eventually become necessary. By the time that TEX's grandchildren are born, it may be that they will bear little resemblance to their ancestor. We can only hope that use of TEX will have become so commercially important that translators of TEX documents to the new generation systems will be developed. An analogy can be found in programming languages: Fortran is a distant ancestor of the Algol family of languages, including Pascal, C, C++, and Ada. An enormous body of important Fortran code exists that cannot possibly be rewritten by hand; public-domain and commercial translators have been developed to convert Fortran code to some of these languages.

While the design of TEX's children is underway, we need to get all TEX systems upgraded to the final versions that Donald Knuth has provided, and we need to agree upon a standard 8-bit TEX font encoding that will permit the exchange of documents that make use of the new features of TEX 3.0. As I noted in my President's message in this issue, this second problem should soon be solved.

References

[1] Elizabeth Barnhart. TEX in the commercial environment—multi-column output. *TUGboat*, 8(2):185, July 1987.

[2] Zeev Becker and Daniel Berry. `triroff`, an adaptation of the device-independent `troff` for formatting tri-directional text. *Electronic Publishing—Origination, Dissemination, and Design*, 2(3):119–142, October 1989.

[3] Nelson H. F. Beebe. TEX and Graphics: The State of the Problem. *Cahiers GUTenberg*, 1(2):13–53, 1989. Presented to: Congrès GUTenberg, Paris, France, 16–17 May 1989.

[4] Michael Ferguson. Multilingual TEX update. *TUGboat*, 7(1):16, March 1986.

[5] Michael Ferguson. A (hopefully) final extension of multilingual TEX. *TUGboat*, 8(2):102, July 1987.

[6] Michael Ferguson. Coordination of non-English use of TEX. *TUGboat*, 11(1):8–9, April 1990.

[7] Michael J. Ferguson. A multilingual TEX. *TUGboat*, 6(2):57, July 1985.

[8] John D. Hobby. A METAFONT-like System with PostScript Output. *TUGboat*, 10(4):505–512, December 1989.

[9] Donald Knuth and Pierre MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat*, 8(1):14, April 1987.

[10] Leslie Lamport. *LATEX—A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, 1985.

[11] Franklin Mark Liang. *Word Hy-phen-ation by Com-pu-ter*. PhD thesis, Stanford University, August 1983.

[12] Frank Mittelbach. E-TEX: Guidelines for future TEX extensions. *TUGboat*, 11(3):337–345, September 1990.

[13] Michael Vulis. VTEX extensions to the TEX language. *TUGboat*, 11(3):429–434, September 1990.

[14] Reinhard Wonneberger and Frank Mittelbach. BIBTEX reconsidered. *TUGboat*, 12(1):xxx–xxx, March 1991.