Keynote Address: The design of TEX and METAFONT: A retrospective

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148
Internet: beebe@math.utah.edu, beebe@acm.org, beebe@computer.org

Abstract

This article looks back at the design of TEX and METAFONT, and analyzes how they were affected by architectures, operating systems, programming languages, and resource limits of the computing world at the time of their creation by a remarkable programmer and human being, Donald E. Knuth. This paper is dedicated to him, with deep gratitude for the continued inspiration and learning that I've received from his software, his writing, and our occasional personal encounters over the last 25+ years.

1 Contents

2 Introduction

More than a quarter century has elapsed since Donald Knuth took his sabbatical year of 1977–78 at Stanford University to tackle the problem of improving the quality of computer-based typesetting of his famous book series, *The Art of Computer Programming* [26, 27, 28, 29, 30, 31].

When the first volume appeared in 1968, most typesetting was still done by the hot lead process, and expert human typographers with decades of experience handled line breaking, page breaking, and page layout. By the mid 1970s, proprietary computer-based analog typesetters had entered the market, and in the view of Donald Knuth, had seriously degraded quality. When the first page proofs of part of the second edition of Volume 2 arrived, he was so disappointed that he wrote [35, p. 5]:

> I didn't know what to do. I had spent 15 years writing those books, but if they were going to look awful I didn't want to write any more. How could I be proud of such a product?

A few months later, he learned of some new devices that used digital techniques to create letter images, and the close connection to the 0's and 1's of computer science led him to think about how he himself might design systems to place characters on a page, and draw the individual characters as a matrix of black and white dots. The sabbatical-year project produced working prototypes of two software programs for that purpose that were described in the book *TEX and METAFONT: New Directions in Typesetting* [32].

The rest is of course history … the digital typesetting project lasted about a decade, produced several more books [36, 37, 38, 39, 40, 34, 35], Ph.D. degrees for Frank Liang [44], John Hobby [16], Michael Plass [48], Lynn Ruggles [49], and Ignacio Zaballa Salelles [57], and had spinoffs in the commercial document-formatting industry and in the first laser printers. TEX, and the LATEX system built on top of it [9, 10, 11, 42, 43, 45], became the standard markup and typesetting system in the computer

science, mathematics, and physics communities, and has been widely used in many other fields.

The purpose of this article is to look back at TEX and METAFONT and examine how they were shaped by the attitudes and computing environment of the time.

3 Computers and people

Now that computers are widely available throughout much of the developed world, and when embedded systems are counted, are more numerous than humans, it is probably difficult for younger people to imagine a world without computers readily at hand. Yet not so long ago, this was not the case.

Until the desktop computers of the 1980s, a 'computer' usually meant a large expensive box, at least as long as an automobile, residing in a climate-controlled machine room with raised flooring, and fed electricity by power cables as thick as your wrist. At many universities, these systems had their own buildings, or at least entire building floors, called Computer Centers. The hardware usually cost hundreds of thousands to millions of dollars (where according to the US Consumer Price Index, a million dollars in 1968 is roughly the same as five million in 2000), and required a full-time professional staff of managers, systems programmers, and operators.

At most computer installations, the costs were passed on to users in the form of charges, such as the US$1500 per hour for CPU time and US$0.50 to open a file that I suffered with as a graduate student earning US$1.50 per hour. At my site, there weren't any disk storage charges, because it was forbidden to store files on disk: they had to reside either on punched cards, or magnetic tape. A couple of years ago, I came across a bill from the early 1980s for a 200MB disk: the device was the size of a washing machine, and cost US$15,000. Today, that amount of storage is about fifty thousand times cheaper.

I have cited these costs to show that, until desktop computers became widespread, it was people who worked for computers, not the reverse. When a two-hour run cost as much as your year's salary, you had to spend a lot of time thinking about your programs, instead of just running them to see if they worked.

When I came to Utah in 1978, the College of Science that I joined had just purchased a DECSYSTEM 20, a medium-sized timesharing computer based on the DEC PDP-10 processor, and the Department of Computer Science bought one too on the same order. Ours ultimately cost about $750,000, and supplied many of the computing needs of the College of Science for more than a dozen years, often supporting 50–100 interactive login sessions. Its total physical memory was just over three megabytes, but we called it three quarters of a megaword. Although computer time was still a chargeable item, we managed to recover costs by getting each Department to contribute a yearly portion of the expenses as a flat fee, so most individual users didn't worry about computer charges.

4 The DEC PDP-10

The PDP-10 ran at least eight or nine different operating systems:

- BBN TENEX,
- Compuserve 4S72,
- DEC TOPS-10 (sometimes jokingly called BOTTOMS-10 by TOPS-20 users),
- DEC TOPS-20 (a modified TENEX affectionately called TWENEX by some users),
- MIT ITS (Incompatible Time Sharing System),
- Stanford WAITS (Westcoast Alternative to ITS),
- Tymshare AUGUST, a modified TOPS-10, and
- Tymshare TYMCOM-X, and on the smaller DECSYSTEM 20/20 model, TYMCOM-XX.

Although the operating systems differed, it was usually possible to move source-code programs among them with few if any changes, and some binaries compiled on TOPS-10 in 1975 still run just fine on TOPS-20 today.

Our machines at Utah both used TOPS-20, but Donald Knuth's work on TEX and METAFONT was done on WAITS. That system was a research operating system, with frequent changes that resulted in bugs, causing many crashes and much downtime. Don told me earlier this year that the O/S was aptly named, since he wrote much of the draft of the TEXbook while he was waiting in the Computer Center for WAITS to come back up.

For about a decade, PDP-10 computers formed the backbone of the Arpanet, which began with just five nodes, at the University of California campuses at Berkeley, Los Angeles, and Santa Barbara, plus SRI (Stanford Research Institute) and Utah, and later evolved into the world-wide Internet [13, p. 48]. PDP-10 machines were adopted by major computer science departments, and hosted or contributed to many important developments, including at least these:

- Bob Metcalf's *Ethernet* [Xerox PARC, Intel, and DEC];
- Vinton Cerf's and Robert Kahn's development of the *Transmission Control Protocol* and the *Internet Protocol* (TCP/IP);

- the MACSYMA [MIT], REDUCE [Utah] and MAPLE [Waterloo] symbolic-algebra languages;
- several dialects of LISP, including MACLISP [MIT] and PSL (Portable Standard Lisp) [Utah];
- the systems-programming language BLISS [DEC and Carnegie-Mellon University (CMU)];
- the shell-scripting language PCL (Programmable Command Language) [DEC and CMU];
- the SAIL (Stanford Artificial Intelligence Language) Algol-family programming language in which TEX and METAFONT were first implemented;
- an excellent compiler for PASCAL [Hamburg/Rutgers/Sandia], the language in which TEX and METAFONT were next implemented;
- Brian Reid's document-formatting and bibliographic system, SCRIBE [CMU], that heavily influenced the design of LATEX and BIBTEX;
- Richard Stallman's extensible and customizable text editor, EMACS [MIT];
- Jay Lepreau's port, PCC20 [Utah], of Steve Johnson's *Portable C Compiler*, PCC [Bell Labs];
- Kok Chen's and Ken Harrenstien's KCC20 native C compiler [SRI];
- Ralph Gorin's SPELL, one of the first sophisticated interactive spelling checkers [Stanford];
- Mark Crispin's mail client, MM, still one of the best around [Stanford];
- Frank da Cruz's transport- and platform-independent interactive and scriptable communications software KERMIT [Columbia].

The PDP-10 and its operating systems is mentioned in about 170 of the now nearly 4000 *Request for Comments* (RFC) documents that informally define the protocols and behavior of the Internet.

The PDP-10 had compilers for ALGOL 60, BASIC, BLISS, C, COBOL 74, FORTH, FORTRAN 66, FORTRAN 77, LISP, PASCAL, SAIL, and SNOBOL, plus three assemblers called MACRO, MIDAS, and FAIL (fast one-pass assembler). A lot of programming was done in assembly code, including most of the operating systems. Indeed, the abstract of the FAIL manual [56] notes:

> Although FAIL uses substantially more main memory than MACRO-10, it assembles typical programs about five times faster. FAIL assembles the entire Stanford time-sharing operating system (two million characters) in less than four minutes of CPU time on a KA-10 processor.

The KA-10 was one of the early PDP-10 models, so such performance was quite impressive. The high-level BLISS language might have been preferred for such work, but it was comparatively expensive to license, and few sites had it. Anyway, Ralph Gorin's book on assembly language and systems programming [12] provided an outstanding resource for programmers.

Document formatting was provided by RUNOFF which shared a common ancestor ROFF with UNIX TROFF. Later, SCRIBE became available, but required an annual license fee, and ran only on the PDP-10, so it too had limited availability, and I refused to use it for that reason.

The PDP-10 had 36-bit words, with five seven-bit ASCII characters stored in each word. This left one bit, the low-order one, left over. It was normally zero, but when set to one, indicated that the preceding five characters were a line number that some editors used, and compilers could report in diagnostics.

Although seven-bit ASCII was the usual PDP-10 text representation, the hardware instruction set had general byte pointer instructions that could reference bytes of any size from 1 to 36 bits, and the KCC20 compiler provided easy access to them in C. For interfacing with 32-bit UNIX and VMS systems, 8-bit bytes were used, with four bits wasted at the low end of each word.

The PDP-10 filesystems recorded the byte count and byte size for every file, so in principle, text-processing software at least could have handled both 7-bit and 8-bit byte sizes. Indeed, Mark Crispin proposed that Unicode could be nicely handled in 9-bit UTF-9 and 18-bit UTF-18 encodings [6]. Alas, most PDP-10 systems were retired before this generality could be widely implemented.

One convenient feature of the PDP-10 operating systems was the ability to define *directory search paths* as values of *logical names*. For example, in TOPS-20, the command

```
@define TEXINPUTS: TEXINPUTS:,
                   ps:<jones.tex.inputs>
```

would add a user's personal subdirectory to the end of the system-wide definition of the search path. A subsequent reference to `texinputs:myfile.tex` was all that it took to locate the file in the search path.

Since the directory search was handled inside the operating system, it was trivially available to all programs, no matter what language they were written in, unlike other operating systems where such searching has to be implemented by each program that requires it.

In addition, a manager could readily change the system-wide definition by a single privileged command:

```
$^Edefine TEXINPUTS: ps:<tex.inputs>,
                     ps:<tex.new>
```

The new definition was immediately available to all users, including those who had included the name `TEXINPUTS:` in their own search paths.

The great convenience of this facility encouraged those who ported TEX and METAFONT to provide something similar. Today, users of the TEXLive distributions are familiar with the `kpathsea` library, which provides an even more powerful mechanism for path searching.

The original PDP-10 instruction set had an 18-bit address field, giving a memory space of $2^{18} = 262\,144$ words, or about 1.25MB. Later designs extended the address space to 30 bits (5GB), but only 23 were ever implemented in DEC hardware, giving a practical limit of 40MB. That was still much more than most customers could afford in 1984 when the PDP-10 product line was terminated, and VAX VMS became the DEC flagship architecture and operating system.

DEC had products based on the KA-10, KI-10, and KL-10 versions of the PDP-10 processor. Later, other companies produced competing systems that ran one or more of the existing operating systems: Foonly (F1, F2, and F3), Systems Concepts (SC-40), Xerox PARC (MAXC) [8], and XKL Systems Corporation (TD-1, TOED-1, and TOAD-1). Some of these implemented up to 27 address bits (128MW, or 576MB). XKL even made a major porting effort of GNU and UNIX utilities, and got the X11 WINDOW SYSTEM running. Ultimately, none enjoyed continued commercial success.

The PDP-10 lives on among hobbyists, thanks to Ken Harrenstien's superb KLH10 simulator [15] with full 30-bit addressing, and the vendor's generosity in providing the operating system, compilers, and utilities for noncommercial use. On a fast modern desktop workstation, TOPS-20 runs several times faster than the original hardware ever did. It has been fun revisiting this environment that was such a leap forward from its predecessors, and I now generally have a TOPS-20 window or two open on my UNIX workstation.

5 Resource limits

The limited memory of the PDP-10 forced many economizations in the design of TEX and METAFONT. Although PASCAL has `new()` and `dispose()` functions for allocating and freeing memory, implementations were allowed to ignore the latter, so Don could not use them. Instead, all memory management is handled by the programs themselves, and sizes of internal tables are fixed at compile time. Table 1 shows the sizes of those tables, then and now. To further economize, many data structures were stored compactly with redundant information elided. Thus, for example, while TEX fonts could have up to 256 characters, there are only 16 different widths and heights allowed, and one of those 16 is required to be zero. Also, although hundreds of text fonts are allowed, only 16 mathematical fonts are supported.

Table 1: TEX table sizes on TOPS-20 in 1984 and in TEXLive on UNIX in 2004, as reported in the `trip` test.

Table	1984	2004	Growth
strings	1819	98002	53.9
string characters	9287	1221682	131.5
memory words	3001	1500022	499.8
control sequences	2100	60000	28.6
font info words	20000	1000000	50.0
fonts	75	2000	26.7
hyphen. exceptions	307	1000	3.3
stack positions (i)	200	5000	25.0
stack positions (n)	40	500	12.5
stack positions (p)	60	6000	100.0
stack positions (b)	500b	200000	400.0
stack positions (s)	600	40000	66.7

Instead of supporting scores of accented characters, TEX expected to compose them dynamically from an accent positioned on a base letter. That in turn meant that words with accented letters could not be hyphenated automatically, an intolerable situation for many European languages. That restriction was finally removed in 1990 with the release of TEX version 3.0 and METAFONT version 2.0, when those programs were extended to fully support 8-bit characters.

The TEX DVI and METAFONT GF and TFM files were designed to be compact binary files that require special software tools to process. In contrast, in UNIX TROFF, these files are generally simple, albeit compact and cryptic, text files to facilitate use of filters in data-processing pipelines. Indeed, the UNIX approach of small-is-beautiful encouraged the use of separate tools for typesetting mathematics, pictures, and tables, instead of the monolithic approach that TEX uses.

Finally, error diagnostics and error recovery reflect past technology and resource limits. Robin Fairbairns remarked in a May 2005 TEXhax list posting:

> Any TEX-based errors are pretty ghastly. This is characteristic of the age in which it was developed, and of the fiendishly feeble machines we had to play with back then. But they're a lot better than the first Algol 68 compiler I played with, which had a single syntax diagnostic "*not a program!*."

6 Choosing a programming language

When Donald Knuth began to think about the problem of designing and implementing a typesetting system and a companion font-creation system, he was faced with the need to select a programming language for the task. We have already summarized what was available on the PDP-10.

COBOL was too horrid to contemplate: imagine writing code in a language with hundreds of reserved words, and such verbose syntax that a simple arithmetic operation and assignment `c = a*b` becomes

```
MULTIPLY A BY B GIVING C.
```

More complex expressions require every subexpression to be given a name and assigned to.

FORTRAN 66 was the only language with any hope of portability to many other systems. However, its lack of recursion, absence of data structures beyond arrays, lack of memory management, deficient control structures, record-oriented I/O, primitive Hollerith strings (`12HHELLO, WORLD`) that could be used only in `DATA` statements and as routine arguments, and its restriction to six-character variable names, made it distinctly unsuitable. Even so, it was later used elsewhere to implement a translation of METAFONT from SAIL for use on Harris computers [46].

PASCAL only became available on the PDP-10 in mid-1982, more than five years after Don began his sabbatical year. We shall return to it in Section 7.

BLISS was an expensive commercial product that was available only on DEC PDP-10, PDP-11, and later, VAX, computers. Although DEC later defined COMMON BLISS to be used across those very different 16-bit, 32-bit, and 36-bit systems, in practice, BLISS exposed too much of the underlying architecture.

LISP would have been attractive and powerful, and in retrospect, would have made TEX and METAFONT far more extensible than they are, because any part of them could have been rewritten in LISP, and they would not have needed to have macro languages at all! Unfortunately, until the advent of COMMON LISP in 1984 [51, 52], and for some time after, the LISP world suffered from having about as many dialects as there were LISP programmers, making it impossible to select a language flavor that worked everywhere.

The only viable approach would have been to write a LISP compiler or interpreter, bringing one back to the original problem of picking a language to write *that* in. The one point in favor of this approach is that LISP is syntactically the simplest of all programming languages, so workable interpreters could be done in a few hundred lines, instead of the 10K to 100K lines that were needed for languages like PASCAL and FORTRAN. However, we have to remember that computer use cost a lot of money, and comparatively few people outside computer science departments had the luxury of ignoring the substantial run-time costs of interpreted languages. A typesetting system is expected to receive a lot of use, and efficiency and fast turnaround are essential.

PDP-10 assembly language had been used for many other programming projects, including the operating system and the three assemblers themselves. However, Don had worked on several different machines since 1959, and he knew that all computers eventually get replaced, often by new ones with radically-different instruction sets, operating systems, and programming languages. Thus, this avenue was not attractive either, since he had to be able to use his typesetting program for all of his future writing.

There was only one viable choice left, and that was SAIL. Although it had an offspring, MAINSAIL (Machine Independent SAIL), that might have been more attractive, that language was not born until 1979, two years after the sabbatical-year project. Figure 1 shows a small sample of SAIL, taken from the METAFONT source file `mfntrp.sai`. A detailed description of the language can be found in the first good book on computer graphics [47, Appendix IV].

The underscore operator in source-code assignments printed as a left arrow in the Stanford variant of ASCII (MIT also had its own flavor), but PDP-10 sites elsewhere just saw it as a plain underscore. However, its use as the assignment operator meant that it could not be used as an extended letter to make compound names more readable, as is now common in many other programming languages.

The left arrow in the Stanford variant of ASCII was not the only unusual character. Table 2 shows graphics assigned to the normally glyphless control characters. The existence of seven Greek letters in the control-character region may explain why TEX's

```
internal saf string array fname[0:2]
# file name, extension, and directory;

internal simp procedure scanfilename
# sets up fname[0:2];
begin integer j,c;
fname[0]_fname[1]_fname[2]_null;
j_0;
while curbuf and chartype[curbuf]=space
    do c_lop(curbuf);
loop    begin c_chartype[curbuf];
        case c of begin
        [pnt] j_1;
        [lbrack] j_2;
        [comma][wxy][rbrack][digit][letter];
        else done
          end;
        fname[j]_fname[j]&lop(curbuf);
        end;
end;
```

Figure 1: Filename scanning in SAIL, formatted as originally written by DEK, except for the movement of comments to separate lines. The square-bracketed names are symbolic integer constants declared earlier in the program.

default text-font layout packs Greek letters into the first ten slots.

Besides being a high-level language with good control and data structures, and recursion, SAIL had the advantage of having a good debugger. Symbolic debuggers are common today, sometimes even with fancy GUI front ends that some users like. In 1977, window systems had not yet made it out of Xerox PARC, and the few interactive debuggers available generally worked at the level of assembly language. Figure 2 shows a small example of a session with the low-level *Dynamic Debugging Tool/Technique*, DDT, that otherwise would have been necessary for debugging most programming languages other than SAIL (COBOL and FORTRAN, and later, PASCAL, also had source-level debuggers).

SAIL had a useful conditional compilation feature, allowing Don to write

```
# changed to ^P^Q when debugging METAFONT;
define DEBUGONLY = ^Pcomment^Q
...
# used when an array is believed to require
# no bounds checks;
define saf = ^Psafe^Q
```

Table 2: The Stanford extended ASCII character set. Character numbers are given in octal.

000	$\cdot$	001	$\downarrow$	002	α	003	β
004	$\wedge$	005	$\neg$	006	ϵ	007	π
010	λ	011	γ	012	δ	013	$\int$
014	$\pm$	015	$\oplus$	016	∞	017	∇
020	$\subset$	021	$\supset$	022	$\cap$	023	$\cup$
024	$\forall$	025	$\exists$	026	$\otimes$	027	$\leftrightarrow$
030		031	$\rightarrow$	032	~	033	$\neq$
034	$\leq$	035	$\geq$	036	$\equiv$	037	$\vee$
040–135 as in standard ASCII							
				136	$\uparrow$	137	$\leftarrow$
140–174 as in standard ASCII							
		175	$\Diamond$	176	}	177	^

```
# used when SAIL can save time implementing
# this procedure;
define simp = ^Psimple^Q

# when debugging, belief turns to disbelief;
DEBUGONLY redefine saf = ^P^Q

# and simplicity dies too;
DEBUGONLY redefine simp = ^P^Q
```

A scan of the SAIL source code for METAFONT shows several other instances of how the implementation language and host computer affected the METAFONT code:

- 19 buffers for disk files;
- no more than 150 characters/line;
- initialization handled by a separate program module to save memory;
- bias of 4 added to case statement index to avoid illegal negative cases;
- character raster allocated dynamically to avoid 128K-word limit on core image;
- magic TENEX-dependent code to allocate buffers between the METAFONT code and the SAIL disk buffers because *there is all this nifty core sitting up in the high seg . . . that is just begging to be used*.

7 Switching programming languages

Donald Knuth initially expected that TEX and METAFONT would be useful primarily for his own books and papers, but other people were soon clamoring for access, and many of them did not have a PDP-10 computer to run them on. The American Mathematical Society was interested in evaluating TEX and METAFONT for its own extensive mathematical publishing activities, but could make an investment in

```
@type hello.pas
program hello(output);
begin
   writeln('Hello, world')
end.

@load hello
PASCAL: HELLO
LINK:   Loading

@ddt
DDT
hello$b   hello+62$b   $$g
$1B>>HELLO/   TDZA 0   $x
   0/   0   0/   0
<SKIP>
HELLO+2/   MOVEM %CCLSW   $x
   0/   0   %CCLSW/   0
HELLO+3/   MOVE %CCLDN   $x
   0/   0   %CCLDN/   0
HELLO+4/   JUMPN HELLO+11   $x
   0/   0   HELLO+11
HELLO+5/   MOVEM 1,%RNNAM   $p

OUTPUT     : tty:
$2B>>HELLO+62/   JRST .MAIN.   $$x
Hello, world
```

Figure 2: Debugging a PASCAL program with DDT. The at signs are the default TOPS-20 command prompt. The dollar signs are the echo of ASCII ESCAPE characters. Breakpoints ($b) are set at the start of the program, and just before the call to the runtime-library file initialization. Execution starts with $$g, proceeds after a breakpoint with $p, steps single instructions with $x, and steps until the next breakpoint with $$x.

switching from the proprietary commercial typesetting system that it was then using *only* if it could be satisfied with the quality, the longevity, and the portability of these new programs.

It was clear that keeping TEX and METAFONT tied to SAIL and the PDP-10 would ultimately doom them to oblivion. It was also evident that some of the program-design decisions, and the early versions of the Computer Modern fonts, did not produce the high quality that their author demanded of himself. Researchers at Xerox PARC has translated the SAIL version of TEX to MESA, but that language ran only on Xerox workstations, which, while full of great ideas, were too expensive ever to make any significant market penetration.

A new implementation language was needed, and in December 1981, when the first source files for the new systems appeared, there was really only one possibility: PASCAL. However, before you rise to this provocation, why not C instead?

UNIX had reached the 16-bit DEC PDP-11 computers at the University of California at Berkeley in 1974. By 1977, researchers there had it running on the new 32-bit DEC VAX, but the C language in which much of UNIX is written was only rarely available outside that environment. Jay Lepreau's PCC20 work was going on in the Computer Science Department at Utah in 1981–82, but it wasn't until about 1983 that TOPS-20 users elsewhere began to get access to it. Our filesystem archives show my first major porting attempt of a C-language UNIX utility to TOPS-20 on 11 February 1983.

PASCAL, a descendant of ALGOL 60 [3], was designed by Niklaus Wirth at ETH in Zürich, Switzerland in 1968. His first attempt at writing a compiler for it in FORTRAN failed, but he then wrote a compiler for a subset of PASCAL in that subset, translated it by hand to assembly language, and was finally able to bootstrap the compiler by getting it to compile itself [54].

Urs Ammann later wrote a completely new compiler [1] in PASCAL for the PASCAL language on the 60-bit CDC 6600 at ETH, a machine class that I myself worked extensively and productively on for nearly four years. That compiler generated machine code directly, instead of producing assembly code, and ran faster, and produced faster code, than Wirth's original bootstrap compiler. Ammann's compiler was the parent of several others, including the one on the PDP-10.

PASCAL is a small language intended for teaching introductory computer programming skills, and Wirth's book with the great title *Algorithms + Data Structures = Programs* [55] is a classic that is still worthy of being studied. However, PASCAL is *not* a language that is suitable for larger projects. A fragment of the language is shown in Figure 3, and much more can be seen in the source code for TEX [37] and METAFONT [39].

PASCAL's flaws are well chronicled in a famous article by Brian Kernighan [17, 18]. That paper was written to record that pain that PASCAL caused in implementing a moderate-sized, but influential, programming project [19]. He wrote in his article:

> PASCAL, at least in its standard form, is just plain not suitable for serious programming. ... This botch [confusion of size and type]

```
PROCEDURE Scanfilename;
 LABEL 30;
BEGIN
 beginname;
 WHILE buffer[curinput.locfield] = 32 DO
  curinput.locfield := curinput.locfield+1;
 WHILE true DO
 BEGIN
  IF (buffer[curinput.locfield] = 59) OR
     (buffer[curinput.locfield] = 37) THEN
     GOTO 30;
  IF NOT morename(buffer[curinput.locfield])
   THEN GOTO 30;
  curinput.locfield := curinput.locfield+1;
 END;
30:
    endname;
END;
```

Figure 3: Filename scanning in PASCAL, after manual prettyprinting. The statements `beginname` and `endname` are calls to procedures without arguments. The magic constants 32, 37, and 59 would normally have been given symbolic names, but this code is output by the TANGLE preprocessor which already replaced those names by their numeric values. The lack of statements to exit loops and return from procedures forces programmers to resort to the infamous `goto` statements, which are required to have predeclared numeric labels in PASCAL.

> is the biggest single problem in PASCAL. ... I feel that it is a mistake to use PASCAL for anything much beyond its original target. In its pure form, PASCAL is a toy language, suitable for teaching but not for real programming.

There is also a good survey of ambiguities and insecurities of the language by Welsh, Sneeringer, and Hoare [53].

Donald Knuth had co-written a compiler for a subset of ALGOL 60 two decades earlier [2], and had written extensively about that language [41, 21, 20, 22, 24, 25]. Moreover, he had developed the fundamental theory of parsing that is used in compilers [23]. He was therefore acutely aware of the limitations of PASCAL, and to enhance portability of TEX and METAFONT, and presciently (see Section 8), to facilitate future translation to other languages, sharply restricted his use of features of that language [37, Part 1].

The botch that Brian Kernighan criticized has to do with the fact that in PASCAL, object sizes are part of their type: if you declare a variable to hold ten characters, then it is illegal to assign a string of any other length to it, and if it appears as a routine argument, then all calls to that routine must pass a string of exactly the correct length.

Donald Knuth's solution to this extremely vexing problem for programs like TEX and METAFONT that mainly deal with streams of input characters was to not use PASCAL directly, but rather, to delegate the problem of character-string management, and other tasks, to a preprocessor, called TANGLE. This tool, and its companion WEAVE, are fundamental for the notion of *literate programming* that he developed during this work [34, 50].

Because PASCAL had mainly been used for small programs, few compilers for that language were prepared to handle programs as large and complex as TEX and METAFONT. Their PASCAL source code produced by TANGLE amounts to about 20,000 lines each when prettyprinted.

Ports of TEX and METAFONT to new systems frequently uncovered compiler bugs or resource limits that had to be fixed before the programs could operate. The 16-bit computers were particularly challenging because of their limited address space, and it was a remarkable achievement when Lance Carnes announced TEX on the HP3000 in 1981 [5], followed not long after by his port to the IBM PC with the wretched 64KB memory segments of the Intel 8086 processor. He later founded a company, *Personal TEX, Inc.* About the same time, David Fuchs completed an independent port to the IBM PC, and that effort was briefly available commercially. David Kellerman and Barry Smith left *Oregon Software*, where they worked on PASCAL compilers, to found the company *Kellerman & Smith* to support TEX in the VAX VMS environment. Barry later started *Blue Sky Research* to support TEX on the Apple MACINTOSH.

8 Switching languages, again

UNIX users had more of a problem getting TEX and METAFONT, because of compiler problems. Pavel Curtis and Howard Trickey first announced a port in 1983 [7], where they noted:

> Unhappily, the PC compiler has more deficiencies than one might wish.

Their project took several months, and ultimately, they had to make several changes and extensions to the PASCAL compiler.

In 1986–1987, Pat Monardo at the University of California, Berkeley, did the UNIX community a great

service when he undertook a translation, partly machine assisted, and partly manual, of TEX from PASCAL to C, the result of which he called COMMON TEX. That work ultimately led to the Web-to-C project to which many people have contributed, and today, virtually all UNIX installations, and indeed, the entire TEXLive distribution for UNIX and Microsoft WINDOWS, is based on the completely automated translation of the master source files of all TEXware and METAFONTware from the Web sources to PASCAL and then to C.

Although we shall not further describe it here, it is worth noting that yet another programming language has since been used to reimplement TEX: Karel Skoupý's work with JAVA [14].

Another interesting project is Achim Blumensath's *ANT: A Typesetting System* [4], where the recursive acronym means *ANT is not TEX*. The first version was done in the LISP dialect SCHEME, and the current version is in OCAML. Input is very similar to TEX markup, and output can be DVI, PostScript, or PDF.

9 Wrapping up

In this article, I have described how architecture, operating systems, programming languages, and resource limits influenced the design of TEX and METAFONT. This analysis is in no way intended to be critical, but instead, offer a historical retrospective that is, I believe, helpful to think about for other widely-used software packages as well.

TEX and METAFONT, and the literate programming system in which they are written, are truly remarkable projects in software engineering. Their flexibility, power, reliability, and stability, and their unfettered availability, have allowed them to be widely used and relied upon in academia, industry, and government. Donald Knuth expects to use them for the rest of his career, and so do many others, including this author. His willingness to expose his programs to public scrutiny by publishing them as books [37, 39], and then to further admit to errors in them [33] in order to learn how we might become better programmers, are traits too seldom found in others.

References

[1] Urs Ammann. On code generation in a PASCAL compiler. *Software—Practice and Experience*, 7(3):391–423, May/June 1977. CODEN SPEXBL. ISSN 0038-0644.

[2] G. A. Bachelor, J. R. H. Dempster, D. E. Knuth, and J. Speroni. SMALGOL-61. *Communications of the Association for Computing Machinery*, 4(11):499–502, November 1961. CODEN CACMA2. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/366813.366843`.

[3] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery*, 6(1):1–17, January 1963. CODEN CACMA2. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/366193.366201`. Edited by Peter Naur. Dedicated to the memory of William Turanski.

[4] Achim Blumensath. ANT: A typesetting system. World-Wide Web document and software, October 24, 2004. URL `http://www-mgi.informatik.rwth-aachen.de/~blume/Download.html`.

[5] Lance Carnes. TEX for the HP3000. *TUGboat*, 2 (3):25–26, November 1981. ISSN 0896-3207.

[6] M. Crispin. RFC 4042: UTF-9 and UTF-18 efficient transformation formats of Unicode, April 2005. URL `ftp://ftp.internic.net/rfc/rfc4042.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc4042.txt`. Status: INFORMATIONAL.

[7] Pavel Curtis and Howard Trickey. Porting TEX to VAX/UNIX. *TUGboat*, 4(1):18–20, April 1983. ISSN 0896-3207.

[8] Edward R. Fiala. MAXC systems. *Computer*, 11(5):57–67, May 1978. CODEN CPTRB4. ISSN 0018-9162. URL `http://research.microsoft.com/~lampson/Systems.html#maxc`.

[9] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54199-8. xxi + 528 pp. LCCN Z253.4.L38 G66 1994.

[10] Michel Goossens and Sebastian Rahtz. *The LATEX Web companion: integrating TEX, HTML, and XML*. Tools and Techniques for Computer Typesetting. Addison-Wesley Longman, Harlow, Essex CM20 2JE, England, 1999. ISBN 0-201-43311-7. xxii + 522 pp. LCCN QA76.76.H94G66 1999. With Eitan M. Gurari and Ross Moore and Robert S. Sutor.

[11] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LATEX Graphics Companion: Illustrating Documents with TEX and PostScript*.

Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1997. ISBN 0-201-85469-4. xxi + 554 pp. LCCN Z253.4.L38G663 1997.

[12] Ralph E. Gorin. *Introduction to DECSYSTEM-20 Assembly Language Programming*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1981. ISBN 0-932376-12-6. xxx + 545 pp. LCCN QA76.8.D17 .G67.

[13] Katie Hafner and Matthew Lyon. *Where wizards stay up late: the origins of the Internet*. Simon and Schuster, New York, NY, USA, 1996. ISBN 0-684-81201-0. 304 pp. LCCN TK5105.875.I57H338 1996.

[14] Hans Hagen. The status quo of the $\mathcal{NTS}$ project. *TUGboat*, 22(1/2):58–66, March 2001. ISSN 0896-3207.

[15] Ken Harrenstien. KLH10 PDP-10 emulator. World-Wide World document and software, 2001. URL `http://klh10.trailing-edge.com/`. This is a highly-portable simulator that allows running TOPS-20 on most modern Unix workstations.

[16] John Douglas Hobby. *Digitized Brush Trajectories*. Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, CA, USA, June 1986. 151 pp. URL `http://wwwlib.umi.com/dissertations/fullcit/8602484`. Also published as report STAN-CS-1070 (1985).

[17] Brian W. Kernighan. Why Pascal is not my favorite programming language. Computer Science Report 100, AT&T Bell Laboratories, Murray Hill, NJ, USA, July 1981. URL `http://cm.bell-labs.com/cm/cs/cstr/100.ps.gz`. Published in [18].

[18] Brian W. Kernighan. Why Pascal is not my favorite programming language. In Alan R. Feuer and Narain Gehani, editors, *Comparing and assessing programming languages: Ada, C, and Pascal*, Prentice-Hall software series, pages 170–186. Prentice-Hall, Englewood Cliffs, NJ, USA, 1984. ISBN 0-13-154840-9 (paperback), 0-13-154857-3 (hard). LCCN QA76.73.A35 C66 1984. See also [17].

[19] Brian W. Kernighan and P. J. Plauger. *Software Tools in Pascal*. Addison-Wesley, Reading, MA, USA, 1981. ISBN 0-201-10342-7. ix + 366 pp. LCCN QA76.6 .K493.

[20] D. E. Knuth, L. L. Bumgarner, D. E. Hamilton, P. Z. Ingerman, M. P. Lietzke, J. N. Merner, and D. T. Ross. A proposal for input-output conventions in ALGOL 60. *Communications of the Association for Computing Machinery*, 7(5):273–283, May 1964. CODEN CACMA2. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/364099.364222`. Russian translation by M. I. Ageev in *Sovremennoe Programmirovanie* **1** (Moscow: Soviet Radio, 1966), 73–107.

[21] Donald E. Knuth. Man or boy? *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 17(??): 7, January 1964. CODEN ALGOBG. ISSN 0084-6198.

[22] Donald E. Knuth. Man or boy? *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 19(7): 8–9, January 1965. CODEN ALGOBG. ISSN 0084-6198.

[23] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, December 1965. CODEN IFCNA4. ISSN 0019-9958. Russian translation by A. A. Muchnik in *Îazyki i Avtomaty*, ed. by A. N. Maslov and É. D. Stotskiĭ (Moscow: Mir, 1975), 9–42. Reprinted in *Great Papers in Computer Science* (1996) [**?**].

[24] Donald E. Knuth. Teaching ALGOL 60. *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 19(??):4–6, January 1965. CODEN ALGOBG. ISSN 0084-6198.

[25] Donald E. Knuth. The remaining trouble spots in ALGOL 60. *Communications of the Association for Computing Machinery*, 10(10): 611–618, October 1967. CODEN CACMA2. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/363717.363743`. Reprinted in E. Horowitz, *Programming Languages: A Grand Tour* (Computer Science Press, 1982), 61–68.

[26] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1968. ISBN 0-201-03803-X. xxi + 634 pp. LCCN QA76.5 .K74. Second printing, revised, July 1969, with page count xxi + 634.

[27] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1969. ISBN 0-201-03802-1. xi + 624 pp. LCCN QA76.5 .K57.

[28] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1971. ISBN 0-201-03802-1. xii + 624 pp. LCCN QA76.5 .K57. Second printing, revised, November 1971.

[29] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1973. ISBN 0-201-03809-9. xxi + 634 pp. LCCN QA76.6 .K641 1973. Second printing, revised, February 1975.

[30] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1973. ISBN 0-201-03803-X. xii + 722 pp. LCCN QA76.5 .K74.

[31] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, March 1975. ISBN 0-201-03803-X. xii + 725 pp. LCCN QA76.5 .K74. Second printing, revised.

[32] Donald E. Knuth. *TEX and METAFONT—New Directions in Typesetting*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1979. ISBN 0-932376-02-9. xi + 201 + 105 pp. LCCN Z253.3 .K58 1979.

[33] Donald E. Knuth. The errors of TEX. Technical Report STAN-CS-88-1223, Stanford University, Department of Computer Science, September 1988. See [33].

[34] Donald E. Knuth. The errors of TEX. *Software—Practice and Experience*, 19(7):607–685, July 1989. CODEN SPEXBL. ISSN 0038-0644. This is an updated version of [**?**]. Reprinted with additions and corrections in [34, pp. 243–339].

[35] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992. ISBN 0-937073-80-6 (paper), 0-937073-81-4 (cloth). xv + 368 pp. LCCN QA76.6.K644.

[36] Donald E. Knuth. *Digital Typography*. CSLI Publications, Stanford, CA, USA, 1999. ISBN 1-57586-011-2 (cloth), 1-57586-010-4 (paperback). xvi + 685 pp. LCCN Z249.3.K59 1998.

[37] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13447-0. ix + 483 pp. LCCN Z253.4.T47 K58 1986.

[38] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13437-3. xv + 594 pp. LCCN Z253.4.T47 K578 1986.

[39] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13445-4. xi + 361 pp. LCCN Z250.8.M46 K58 1986.

[40] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13438-1. xv + 560 pp. LCCN Z250.8.M46 K578 1986.

[41] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2. xv + 588 pp. LCCN Z250.8.M46 K574 1986.

[42] Donald E. Knuth and Jack N. Merner. ALGOL 60 confidential. *Communications of the Association for Computing Machinery*, 4(6):268–272, June 1961. CODEN CACMA2. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/366573.366599`.

[43] Leslie Lamport. *LATEX—A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985. ISBN 0-201-15790-X. xiv + 242 pp. LCCN Z253.4.L38 L35 1986.

[44] Leslie Lamport. *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994. ISBN 0-201-52983-1. xvi + 272 pp. LCCN Z253.4.L38L35 1994.

[45] Phillip Laplante, editor. *Great papers in computer science*. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996. ISBN 0-314-06365-X (paperback), 0-07-031112-4 (hardcover). iv + 717 pp. LCCN QA76 .G686 1996. URL `http://bit.csc.lsu.edu/~chen/GreatPapers.html`.

[46] Franklin Mark Liang. Word hy-phen-a-tion by com-pu-ter. Technical Report STAN-CS-83-977, Stanford University, Stanford, CA, USA, August 1983.

[47] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004. ISBN 0-201-36299-6. xxvii + 1090 pp. LCCN Z253.4.L38 G66 2004.

[48] Sao Khai Mong. A Fortran version of METAFONT. *TUGboat*, 3(2):25, October 1982. ISSN 0896-3207.

[49] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill Computer Science Series, Editors: Richard W. Hamming and Edward A. Feigenbaum. McGraw-Hill, New York, NY, USA, 1973. ISBN 0-07-046337-9. xxviii + 607 pp. LCCN T385 .N48.

[50] Michael F. Plass. *Optimal pagination techniques for automatic typesetting systems*. Thesis (ph. d.), Stanford University, Stanford, CA, USA, 1981. vi + 72 pp.

[51] Lynn Elizabeth Ruggles. *Paragon, an interactive, extensible, environment for typeface design*. Ph.D. dissertation, University of Massachusetts Amherst, Amherst, MA, USA, 1987. 192 pp. URL `http://wwwlib.umi.com/dissertations/fullcit/8805968`.

[52] E. Wayne Sewell. *Weaving a Program: Literate Programming in WEB*. Van Nostrand Reinhold, New York, NY, USA, 1989. ISBN 0-442-31946-0. xx + 556 pp. LCCN QA76.73.W24 S491 1989.

[53] Guy L. Steele Jr. *Common Lisp—The Language*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1984. ISBN 0-932376-41-X. xii + 465 pp. LCCN QA76.73.L23 S73 1984. US$22.00.

[54] Guy L. Steele Jr. *Common Lisp—The Language*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, second edition, 1990. ISBN 1-55558-041-6 (paperback), 1-55558-042-4 (hardcover), 0-13-152414-3 (Prentice-Hall). xxiii + 1029 pp. LCCN QA76.73.L23 S73 1990. See also [51].

[55] J. Welsh, W. J. Sneeringer, and C. A. R. Hoare. Ambiguities and insecurities in Pascal. *Software—Practice and Experience*, 7(6):685–696, November/December 1977. CODEN SPEXBL. ISSN 0038-0644.

[56] Niklaus Wirth. The design of a PASCAL compiler. *Software—Practice and Experience*, 1(4): 309–333, October/December 1971. CODEN SPEXBL. ISSN 0038-0644.

[57] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, NJ, USA, 1976. ISBN 0-13-022418-9. xvii + 366 pp. LCCN QA76.6 .W561.

[58] F. H. G. Wright II and R. E. Gorin. *FAIL*. Computer Science Department, Stanford University, Stanford, CA, USA, May 1974. Stanford Artificial Intelligence Laboratory Memo AIM-226 and Computer Science Department Report STAN-CS-74-407.

[59] Ignacio Andres Zaballa Salelles. *Interfacing with graphics objects*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, USA, December 1982. 146 pp.