Taurida National Vernadsky University

THREE PERSONS WHO CHANGED THE WORLD

Katsitadze O. G.
a first year student
Faculty of Mathematics and
Computer Science

Scientific Adviser:

Anokhina N. P.
Foreign Languages Department

Simferopol – 2004

Part 1
DONALD E. KNUTH

1.0. Background

Donald Ervin Knuth is considered one of the greatest living mathematicians. He specializes in Computer Science. To him is attributed promoting Computer Science from "craft" to science. His many scientific monographs include *The Art of Computer Programming*, which is now considered "the bible" of every computer scientist.

Another major achievement of Donald Knuth is TEX typesetting engine for producing high-quality scientific documents (and its by-product, the so-called "literate programming" concept and tools). It is not surprising therefore that he is versed in typesetting, and his many books on the subject, among which is *Computers and Typesetting* series, cover the typesetting tools he developed and general typesetting and font design principles.

1.1. Biography

education

Donald E. Knuth was born on January 10, 1938 in Milwaukee, Wisconsin. He studied mathematics as an undergraduate at Case Institute of Technology, where he also wrote

unprec.'d step

software at the Computing Center. The Case faculty took the unprecedented step of awarding him a Master's degree together with the B.S. he received in 1960. After graduate studies at California Institute of Technology, he received a Ph.D. in Mathematics in 1963

employment

and then remained on the mathematics faculty. Throughout this period he continued to be involved with software development, serving as consultant to Burroughs Corporation from 1960–1968 and as editor of Programming Languages for ACM publications from 1964–1967.

professor

He joined Stanford University as Professor of Computer Science in 1968, and was appointed to Stanford's first endowed chair in computer science nine years later. As a university professor he introduced a variety of new courses into the curriculum, notably Data Structures and Concrete Mathematics. In 1993 he became Professor Emeritus of The Art of Computer Programming. He has supervised the dissertations of 28 students.

authoring

Knuth began in 1962 to prepare textbooks about programming techniques, and this work evolved into a projected seven-volume series entitled *The Art of Computer Programming*. Volumes 1–3 first appeared in 1968, 1969, and 1973. Having revised these three in 1997, he is now working full time on the remaining volumes. Approximately one million copies have already been printed, including translations into six languages. He took

TEX
METAFONT
`WEB`, `CWEB`

ten years off from this project to work on digital typography, developing the TEX system for document preparation and the METAFONT system for alphabet design. Noteworthy by-products of those activities were the `WEB` and `CWEB` languages for structured documentation, and the accompanying methodology of Literate Programming. TEX is now used to produce most of the world's scientific literature in physics and mathematics.

research His research papers have been instrumental in establishing several subareas of computer science and software engineering: LR(k) parsing; attribute grammars; the Knuth-Bendix algorithm for axiomatic reasoning; empirical studies of user programs and profiles; analysis of algorithms. In general, his works have been directed toward the search for a proper balance between theory and practice.

awards Professor Knuth received the ACM Turing Award in 1974 and became a Fellow of the British Computer Society in 1980, an Honorary Member of the IEEE in 1982. He is a member of the American Academy of Arts and Sciences, the National Academy of Sciences, the National Academy of Engineering, and a foreign associate of l'Academie des Sciences (Paris) and Det Norske Videnskaps-Akademi (Oslo). He holds five patents and has published approximately 160 papers in addition to his 19 books. He received the Medal of Science from President Carter in 1979, the American Mathematical Society's Steele Prize for expository writing in 1986, the New York Academy of Sciences Award in 1987, the J. D. Warnier Prize for software methodology in 1989, the Adelsköld Medal from the Swedish Academy of Sciences in 1994, the Harvey Prize from the Technion in 1995, and the Kyoto Prize for advanced technology in 1996. He was a charter recipient of the IEEE Computer Pioneer Award in 1982, after having received the IEEE Computer Society's W. Wallace McDowell Award in 1980; he received the IEEE's John von Neumann Medal in 1995. He holds honorary doctorates from Oxford University, the University of Paris, St. Petersburg University, and more than a dozen colleges and universities in America.

personal Professor Knuth lives on the Stanford campus with his wife, Jill. They have two children, John and Jennifer. Music is his main avocation.

1.2. *The Art of Computer Programming*

names The book consists of 7 volumes, of which only first three are published, Volume 4 is in preparation, and Volumes 5, 6, and 7 are still only projected. Names of the volumes are:

- Volume 1 *Fundamental Algorithms*
- Volume 2 *Seminumerical Algorithms*
- Volume 3 *Sorting and Searching*
- Volume 4 *Combinatorial Algorithms*, in preparation. If all goes as planned, Volume 4 will be ready in the year 2007.
- Volume 5 *Syntactic Algorithms*, estimated to be ready in 2010.

After Volumes 1–5 are done, Donald Knuth plans to publish Volume 6 (the theory of context-free languages) and Volume 7 (compiler techniques).

history Back in the 1960's Donald Knuth started out to write a book in 12 chapters entitled *The Art of Computer Programming*. By 1967 the plan had expanded to a seven-volume work (still only 12 chapters). The first three volumes had been published by 1973. Since then the first two volumes have been through 2nd editions and now 3rd editions. The outline for Volume 4 (just two of the original 12 chapters) has expanded to three subvolumes, but we are still waiting to see it. The chance of volumes 6 & 7 ever being written seems increasingly remote.

Amer. Scientist

At the end of 1999, these books were named among the best twelve scientific monographs of the century by *American Scientist*, along with: Dirac on quantum mechanics, Einstein on relativity, Mandelbrot on fractals, Pauling on the chemical bond, Russell and Whitehead on foundations of mathematics, von Neumann and Morgenstern on game theory, Wiener on cybernetics, Woodward and Hoffmann on orbital symmetry, Feynman on quantum electrodynamics, Smith on the search for structure, and Einstein's collected papers.

detailed

The book is not about computer programming in the broad sense, but about the algorithms and methods which lie at the heart of most computer systems. *Fundamental Algorithms* contains background information for the series. Chapter 1 provides mathematical preliminaries and basic programming concepts, along with an introduction to the `MIX` assembly language, used throughout for implementations. Chapter 2 covers simple information structures: lists, trees, and related data structures.

The two chapters in *Seminumerical Algorithms* cover pseudo-random numbers—their generation and statistical testing—and numerical computation—doing arithmetic with floating point numbers, rationals, and polynomials.

Almost everyone who has ever programmed has written a bubble sort at some point, but the full complexities of sorting algorithms are another story entirely. After an introduction to the mathematics of permutations, *Sorting and Searching* presents and analyzes an extensive array of algorithms for sorting in memory (insertion, exchange, selection, merging, and distribution algorithms), sorting on secondary storage, and searching.

general

The Art of Computer Programming is not a work for everyone, not even for all programmers. It will be a valuable reference for those working on the implementation and optimization of key algorithms and data structures, but the more mathematically inclined will dip into it simply for pleasure. Knuth himself clearly enjoys the subtleties of the mathematics as much as anything; he writes at one point:

quote on sorting

> "Even if sorting were almost useless, there would be plenty of rewarding reasons for studying it anyway! The ingenious algorithms that have been discovered show that sorting is an extremely interesting topic to explore in its own right. Many fascinating unsolved problems remain in this area, as well as quite a few solved ones." [*Sorting and Searching*, page 3]

and he provides some gloriously learned historical tidbits and mathematical digressions. The mathematics *is* heavy going in places, but the more difficult sections are marked and the material is laid out in such a way that those seeking algorithms to implement and performance analyses can skip the proofs and derivations and the more esoteric material.

exercises

Exercises are liberally provided, along with proper answers, which take up around a quarter of each volume. The exercises are carefully graded in difficulty on a scale from 0 to 50, and range from trivial tests of definitions to unsolved research problems. Reading *The Art of Computer Programming* is a serious enough undertaking in itself, but anyone who succeeds in doing all the exercises will have earned themselves several doctorates.

1.3. *Concrete Mathematics*

This book was written by Donald Knuth in co-authorship with Ronald L. Graham and Oren Patashnik.

expl. 'concrete' What is "concrete" math, as opposed to other types of math? The authors explain that the title comes from the blending of CONtinuous and disCRETE math, two branches of math that many seem to like to keep asunder, though each occurs in the foundation of the other. The topics in the book, such as sums, generating functions, and number theory, are actually standard discrete math topics; however, the treatment in this text shows the inherent continuous (read: calculus) undergirding of the topics. Without calculus, generating functions would not have come to mind and their tremendous power could not be put to use in figuring out series.

description This is a serious math book for those who are willing to dot every i and cross every t. Unlike most math texts (esp. graduate math texts), nothing is omitted along the way. Notation is explained (*very* important), common pitfalls are pointed out (as opposed to the usual way students come across them—by getting back bleeding exams), and what is important and what is *not* as important are indicated.

To someone who has been through the rigors of math grad school, this book is a delight to read; to those who have not, they must keep in mind that this is a serious text and must be prepared to do some real work. Very bright high school students have gotten through this text with little difficulty.

exercises Some of the exercises in the book are serious research topics. They don't necessarily tell you that when they give you the problem; if you've worked on the problem for a week, you should turn to the answers in the back to check that there really is a solution.

The formulas from this book can actually come in handy "in real life," especially if one has to use math a lot.

1.4. TEX

Contemporary history of typesetting

old way Until about twenty years ago, typesetting was virtually ignored by the vast majority of mathematicians, scientists, and scholars in general: manuscripts were prepared using a typewriter, the more esoteric symbols (which meant almost all symbols for mathematicians) were laboriously inserted by hand, and the whole was then simply dispatched to the publisher. Some time later galleys would be returned, emendations noted in the margin, and once again the whole would be sent to the publisher. A similar but shorter cycle was probably repeated for the page proofs, and finally the author's intentions appeared in final form in the finished book. At no point did the author and the typesetter communicate directly, and indeed the former was almost certainly virtually unaware of the latter's existence.

types.'s probl. The typesetter, however, was only too aware of the author: mathematical copy is traditionally referred to as 'penalty copy' in the printing trade, since it is notoriously

difficult to set correctly. In the time that his colleague could set ten pages of straight text, the mathematical typesetter was barely able to accomplish a single page, and even when set he knew that there was every possibility that it would have to be reset more than once, since mathematicians are only too keen to invent new symbols of their own when no existing symbol seems entirely appropriate. And since the typesetter would never have encountered such a symbol before, he would (quite reasonably) assume that it was simply a badly drawn version of a symbol with which he *was* familiar, and substitute the latter...

daisy-wheel

Needless to say, some of the more aware authors began experimenting with computer technology as soon as it became generally accessible, and for a while the academic world seemed convinced that if it were possible to get just a couple more symbols onto the daisy wheel of a Diablo printer, all would become possible: there were even specialist companies who would re-mold a daisy wheel, replacing an apparently unwanted glyph with one which its owner deemed indispensable. Of course, the approach was doomed to failure: one can no more set mathematics with a fixed set of 144 glyphs than can one with a set of 128, and despite the best efforts of all concerned, the daisy-wheel printer was soon consigned to the scrap bin.

dot-matrix

In parallel with this, the dot-matrix printer manufacturers first began to have a significant impact. With a 7×5 dot matrix, there are potentially

$$\sum_{i=0}^{35} \binom{35}{i} = 2^{35}$$

different characters (a very large number indeed!), but unfortunately a number of these are virtually indistinguishable: a single dot at coordinates $(4, 3)$ looks astonishingly like another single dot at coordinates $(4, 4)$ to even the most astute reader (there are something like 33,034,338,305 *distinct* characters, as opposed to a total of 34,359,738,368 characters, where a character is regarded as *distinct* if it's not simply the result of sliding another character horizontally, vertically, or both; this figure is based on an analysis by Dr. Warren Dicks of the Autonomous University of Barcelona). Furthermore, the print quality of a 7×5 dot-matrix printer is so appallingly bad that no attempt should ever be made to set a book using one—unfortunately this well-meant advice was seldom heeded at the time.

ROFF

Of course, in order to exploit these technological revolutions, suitable software had to be written, and the UNIX world in particular decided to standardize on ROFF and its derivatives: NROFF, TROFF, and finally DITROFF all made their mark. Unfortunately none of the ROFF derivatives ever directly supported the typesetting of mathematics, and so adjunct programs such as EQN and TBL had to be used to add mathematical functionality. There were also commercial systems, used to set publications such as the *Transactions of the American Mathematical Society*, but these were both expensive and arcane, using a rather non-mnemonic syntax to represent the possible mathematical constructions.

commercial

Fortunately (as is absolutely clear in retrospect), at least one eminent mathematician believed that something better not only could, but *should*, be created; and being not only a mathematician but a computer scientist, he decided to create it. His name was Knuth, and his creation was TEX.

TEX comes into play

happy conincid. Yet had it not been for a happy coincidence, TEX might never have been born. At the
TAOCP time, Knuth was working on his *opus magnum*, a seven-book series entitled *The Art of*
2nd edition *Computer Programming*, and by 1977 the popularity of the early volumes of this series had proved so great that Volume 2 had already run to a second edition. Unfortunately, or
hot-lead fortunately as it turned out later, the timing of this was such that whilst the first edition had been set using traditional hot-lead technology, the second edition was produced using
1st phototype one of the first phototypesetters [an aside to readers: throughout this paper the term *typesetter* is used to mean both the *person* performing the task of setting type, and the *equipment* used to achieve that end; it is hopefully always clear from the context which of these two meanings is to be inferred, since there is no other word which could easily and felicitously be substituted for either of these usages]. And whilst the new phototypesetter was more than capable *in theory* of achieving results as good as, if not better than, the traditional hot-lead device used previously, the results in practice left a great deal to be
start TEX devel. desired. Knuth, as a mathematician and computer scientist, was convinced that the fault lay not in the technology but in the software used to drive it, and he decided that rather than see his life's work appear in second-rate format, he would devote a short portion of his professional life to developing a suite of software which *would* exploit the full potential of the phototypesetter. Little did he know when he took this brave decision that it was to take not the anticipated one year but at least ten, although he most certainly had a demonstrably working version within his anticipated time frame.

1st ref. to TEX The first published reference to TEX is probably *Mathematical Typography*, published as report *STAN–CS–78–648* by the Computer Science Department of Stanford University; in the bibliography to this, Knuth gives the definitive reference as being *Tau Epsilon Chi*, a system for technical text which was at the time "in preparation." It is now sadly out of print. For those interested in the subject, the former paper makes fascinating reading, and the bibliography alone makes it a more than worthwhile acquisition; it was reproduced in the *Bulletin of the American Mathematical Society*, in which form it should still be available.

typical TEX was both typical and atypical of programs of its era: it was typical in that it was completely script-oriented, predating as it did any widely used graphical user interface;
atypical it was atypical in that it was a completely programmable *macro* programming language, in which there were no reserved words, and in which even individual characters could change their semantics on the fly. Thus a TEX document consisted both of the text to be typeset and the commands to accomplish that typesetting, and only TEX itself could unambiguously determine whether any particular element of the document was to be interpreted as 'program' or 'data'.

standard in SU Despite being created primarily in order to accomplish one particular end—the typesetting of Volume 2 of *The Art of Computer Programming*—TEX rapidly took on a life of its own, and soon became the *de facto* standard for typesetting within much of Stanford
TUG University. Before long its fame had spread, and by 1980 the TEX Users Group had sprung

into existence, with members of the Steering Committee drawn from far beyond the restricted domain of Stanford faculty. The American Mathematical Society were represented
trademark prot. on that Committee, and liaison between the AMS and Knuth was very close: Knuth assigned the TEX logo to the AMS who then applied for trademark protection to prevent it being used to describe any unauthorized modification of TEX—unfortunately this application was rejected because of a prior registration of TEX (sic) by Honeywell, but despite this lack of formal registration, Knuth's high profile and high standing ensure that the TEX logo (or its non-typeset equivalent, `TeX`) is universally recognized and respected.

Evolution of TEX. `WEB`, `TANGLE`, and `WEAVE`

reimplement. Within a couple of years, it became clear that the initial implementation of TEX left something to be desired, both in terms of functionality and in terms of portability, and Knuth set out to redress both by reimplementing TEX from scratch. This time he decided to eschew SAIL ('Stanford Artificial Intelligence Language') as the language of implementation, and instead to adopt the far more widely available programming language Pascal. To further increase its portability, he adopted only a strict subset of Pascal, encompassing only those features which he was confident could be found (or easily emulated) on all Pascal
literate progr. implementations; but he also decided to take this opportunity to render the program in a form which he termed 'literate': that is, he wanted people to be able to *read* the source of TEX in the same way that they might read a book, and to therefore be able to benefit by being exposed to a major piece of software engineering presented in a highly literate manner. Once again Knuth decided that there were no adequate tools available for this, and once again he digressed from the main project by breaking off to design and implement the concept of a `WEB` program, together with its two adjunct programs `TANGLE` and `WEAVE`.

`WEB` A `WEB` program consists of a highly stylized dialect of Pascal, interspersed by lengthy comments describing the purpose and function of every element and module of the program (Knuth would probably deny this, and say that a `WEB` program consists of a highly elaborate description of the workings of the program, interspersed by occasional fragments of Pascal which implement that functionality: and he would almost certainly be right!). By permitting the elements of a Pascal program to be presented in arbitrary order (as opposed to the strict order of presentation required by the Pascal standard), `WEB` allows the programmer the opportunity to present the elements of a program in a natural and logical order, as opposed to the artificial order imposed by the Pascal design criterion of 'efficient compilability'; it is then the task of `TANGLE` to paste together these fragments in the order required by Pascal, and the task of `WEAVE` to bring together both the program fragments and the comment fragments into a form which can immediately be typeset by TEX.

self-referential Thus for the first time TEX became self-referential: in order to be able to produce the Pascal code from the `WEB` source, one needed a working version of `TANGLE`; to be able to produce a literate listing of the `WEB` source, one needed a working copy of `WEAVE`; but both `TANGLE` and `WEAVE` are themselves written in `WEB`, so to produce a working `TANGLE`

one needs a working TANGLE, and so *ad infinitum*. Of course 'bootstrapping' (as the technique is generally termed) is well understood in the Computer Science world, and it was estimated that the task of 'hand compiling' TANGLE from the WEB source was well within the competence of 'the average implementor'; however, there are stories of people suffering many pains attempting this bootstrapping for themselves...

During the reimplementation, Knuth rewrote almost the complete TEX program: he had learned much about its limitations during the first couple of years of use, and by 1982 a completely rewritten TEX had emerged. This version of TEX (often referred to as TEX82, to differentiate it from the earlier version which analogously became known as TEX78) was rapidly ported to a wide range of machines, and is quite possibly the most widely available program in the world today, being available on every class of system from the smallest PC to the largest supercomputer. Its almost universal acceptance as *the* standard package for computer typesetting is almost certainly the result of a large set of very positive attributes: the source of the program, and the vast majority of implementations, are available either free of charge or at a modest cost which covers no more than the media on which they are supplied; the program is virtually bug-free, a claim which Knuth backed up until very recently by offering a check for every bug found, the value of the check doubling each year since the scheme's inception (he still offers a check, but the value no longer doubles, since he estimated that before too long it might exceed the total Federal reserves...); the program is highly stable (there were virtually no major changes during the period 1982–90, and similarly there have been virtually no changes at all since 1990, nor will there be at any point in the future); and there exists a large community of TEX users organized into TUG (TEX Users Group, see below), so any real problems resulting from a lack of experience with TEX can be rapidly resolved by a message to any one of a number of TEX-related mailing lists and news groups.

TEX82
TEX78
universal accept.
positive attribs.
check for a bug
community assist.

So, during the 1980's, TEX emerged as *the* standard package for computer typesetting: it was available on almost every conceivable system, device drivers were written for everything from dot-matrix printers to 2400dpi phototypesetters (but *not* daisy-wheel printers!), and an ever-increasing number of publications appeared which were either typeset using TEX, or were about TEX, or both. Many scientific journals adopted it (or one of its derivatives such as LATEX, which may be thought of as a somewhat restrictive but more user-friendly 'front end' to TEX) as the standard format in which papers were to be prepared. Since an author could very easily proof a paper using a local implementation of TEX, and since TEX was guaranteed to produce identical results no matter on which system it was run, the number of iterations between author and publisher was reduced to the bare minimum, so everybody was happy. And since TEX has been designed by a mathematician, and since a part of its objective had been to allow mathematics to be typeset almost as easily as running text, its take-up by the mathematical community was if anything even faster than its take-up by the scientific and academic communities in general.

standard package
device drivers
scient. journals
identical results
math-oriented

Example of typesetting a math formula with TEX

To give a simple example of why TEX is ideally suited to the typesetting of mathematics, consider the following set of equations:

$$\begin{aligned}\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy \\ &= \int_0^{2\pi}\int_0^{\infty} e^{-r^2}\,r\,dr\,d\theta \\ &= \int_0^{2\pi}\left(-\frac{e^{-r^2}}{2}\Bigg|_{r=0}^{r=\infty}\right)d\theta \\ &= \pi. \end{aligned} \tag{11}$$

by hand A mathematician writing this by hand would almost certainly start with the left-most element of the first line, proceed from left to right, and alternate between baseline, subscript *WYSIWYG* and superscript elements as logic dictated; a pure WYSIWYG ('What you see is what you get') word processor, on the other hand, would require the typist to analyze each row of the equations into horizontal strata (thus the top stratum might contain only ∞, 2, ∞, and ∞, for example) and to enter these stratum by stratum; since, in general, WYSIWYG systems do not automatically displace preceding or following lines of text horizontally when an intervening line is shortened or lengthened, the correction of such equations is *adv. WYSIWYG* tedious and error-prone in the extreme. More recent, WYSIWYG-like, systems require a different approach in which the author has to enter the formula in the order dictated by its parse-tree; needless to say, this approach too demands more of the author than should reasonably be expected.

TEX's way TEX allows the mathematician to enter the formulas in the most natural manner, starting at the left and finishing at the right; alignment is automatically maintained if insertions or deletions are made, and even the horizontal alignment of the four primary = signs is performed automatically, virtually regardless of the length of individual left or right elements. To clarify this, here is the exact TEX source which was used to set the table:

```
$$
\eqalignno
  {\biggl(
      \int_{-\infty}^\infty e^{-x^2}\,dx
   \biggr)^2
  &=\int_{-\infty}^\infty
    \int_{-\infty}^\infty
        e^{-(x^2+y^2)}\,dx\,dy \cr
  &=\int_0^{2\pi}\int_0^\infty
        e^{-r^2}r\,dr\,d\theta \cr
  &=\int_0^{2\pi}
    \biggl(
      -{e^{-r^2}\over 2}
```

```
            \bigg \vert_{r=0}^{r=\infty}\,
        \biggr)
        \,d\theta \cr
      &=\pi.&(11)\cr
    }
    $$
```

ignores spaces It is worth noting that TEX completely ignores any spaces in mathematical text, since the rules for typesetting mathematics are complex, and cannot be expected to be understood by mere mathematicians! Thus the layout of the equations above is simply for the convenience of the author, and is completely ignored by TEX, which is far more *special chars* concerned by special characters such as dollars, backslashes, braces, underscores, carets and ampersands. And whilst each of these characters has a distinct meaning to TEX (a dollar symbol, for example, both introduces and terminates a stretch of mathematical text), that *overr. semantics* meaning may at any time be overridden, and either assigned to a different character or, if not needed, turned of completely. So, for example, if some particular computer lacked a backslash key, it would be trivial to assign the semantics of backslash to some other key (say, yen, if a Japanese keyboard were to be used).

mnemonic Furthermore, it can be seen that TEX is highly mnemonic in its choice of control sequences ('commands', preceded by a backslash); to pick out just a few examples, \int represents an integral sign, \infty an infinity, \exp the exp operator (representing the *logical order* exponential e), and so on. Compound subscripts and superscripts are presented in logical order, rather than in order of their appearance vertically on the page; and facilities are provided for the author to give TEX hints about the logical structure of the expression, so that (for example), \, is used to set off differentials such as $d\theta$ from the preceding text by a little extra white space, thereby improving both the appearance and the legibility of the expression.

TEX attractions Thus the attraction of TEX for mathematicians is clear: a highly logical markup language, capable of being entered from any keyboard; access to a very wide range of mathematical symbols; professional standards of layout; widespread acceptability by journals; and the ability to proof on anything from a dot-matrix printer to a 600dpi laser printer. Add to this the now universal ability to preview the document on the computer screen (something the early advocates of TEX could only dream of), and it is hard to explain why any mathematician with access to a computer would *not* typeset his papers using TEX!

Internationalization of TEX. Reimplementations of TEX

However, use of TEX is restricted neither to mathematicians nor to North Americans, and at the TEX User Group conference in 1989, an influential and voluble group of European TEX users ganged up on Knuth and succeeded in convincing him that, despite his assertion on the previous day of the conference that the development of TEX was finished, there were features missing from the current implementation which made TEX entirely useless to the majority of the world, since whilst it behaved perfectly in unaccented languages, it was grossly deficient for typesetting any language which made more than occasional use of

diacritics. And Knuth, recognizing the validity of this argument, agreed that something had to be done.

TEX3

The result of all this was TEX3: TEX82 became known simply as TEX2, and TEX3 became the One True TEX. In practice, this just didn't happen: those who had no need for the extended diacritic support offered by TEX3 simply continued to use TEX2, and for quite a while TEX macro writers had to write very defensive code which first checked the environment before making any assumptions about (for example) the number of distinct characters with which TEX could internally deal (this was 128 prior to TEX3, and 256 thereafter). With the release of TEX3, Knuth made it *absolutely* clear that this really did represent the end of the TEX evolutionary line: he had better things to do with his time, and TEX was now frozen (modulo any essential bug fixes, which he undertook to continue to make if and only if it could be shown that their fixing was essential). Furthermore he made it equally plain that TEX could *not* be further evolved by anyone else: he wished to leave for his children, and for his children's children, and for all perpetuity, TEX as his creation, and not as his-creation-as-modified-by-someone-else.

TEX is frozen

In general, the TEX world took this in good part: Knuth is enormously highly respected by those who use TEX, and there were very few who advocated ignoring his wishes and who were prepared to suggest modifying TEX. But there were also a quite significant number of TEX users who felt that if TEX did *not* evolve, then it would simply die. Not because of any fundamental deficiencies in TEX—it is generally accepted that there are very few—but because the world had moved on since 1978, and whilst a script-driven language might have been state-of-the-art then, it most certainly was not state-of-the-art now. Furthermore, despite increasing the number of distinct internal characters from 128 to 256, Knuth had done little if anything to enhance TEX to deal with Asian languages, in which the number of distinct characters may be measured in thousands if not in tens of thousands. And finally, there were those who felt that there were *some* areas in which a very significant increase in functionality could be gained (particularly from the perspective of the macro programmer, who is also known as a 'format writer' when the suite of macros provides a complete functional system in its own right) with relatively little investment in terms of modifying TEX.

would die

Asian languages

cheap improvem.

successors

Blue Sky

3B2

64-bit version

NTS's e-TEX

The implementation of these ideas probably represents the leading edge of TEX technology today: companies such as Blue Sky have produced instantaneous/incremental TEX interpreters, which are capable of displaying the effects of a change to the source code of a TEX document in real time; Advent Publishing have produced 3B2, which allows both a graphical and a textual specification of a layout, automatically updating one to reflect changes in the other; John Plaice and Yannis Haralambous have implemented a 64-bit version of TEX which uses Unicode internally; and the NTS group, where NTS stands for 'New Typesetting System', have produced a completely compatible successor to TEX, called e-TEX, which adds functionality without compromising compatibility (the NTS group also wish to re-implement TEX from scratch, using a modern rapid-prototyping language such as Prolog or CLOS, the idea being to allow rapid experimentation with alternative typesetting algorithms or paradigms). Whether or not any of these ideas will catch on remains

to be seen, although among Apple Macintosh *aficionados* Classic Textures (the Blue Sky product referred to above) is already highly thought of. One fundamental question is that of stability: since one of the great strengths of TEX is its stability, how will the world feel about systems which encompass TEX but which are specifically intended to remain evolutionary and responsive, rather than fossilized and unyielding? Only time will tell.

stability issue

TEX is honored

What is perhaps worth noting is that all of these projects have ensured that Knuth's wishes are honored not only in the letter but in the spirit: none seeks to call itself TEX (indeed, that of John Plaice and Yannis Haralambous is called Omega, which could never be confused with TEX), yet all acknowledge the debt which they owe to Knuth and to TEX: without them, none of these other projects would ever have seen the light of day.

TUG and *TUGboat*

TUG

There are an enormous number of TEX users throughout the world, who have banded together to form the TEX Users Group (TUG), in order to exchange information about common problems and solutions. Most of TUG's members are only too keen to pass on their expertise to any who need it, so any real problems resulting from a lack of experience with TEX can be rapidly resolved by a message to any one of a number of TEX-related mailing lists and news groups. Even those without network access are not cut off, as the TUG office offers telephone support from 03:00 in the morning until late in the evening—a service which is *not* restricted to members of TUG.

phone support

TUGboat

A newsletter/journal called *TUGboat* has been published since 1980, featuring articles about all aspects of TEX and METAFONT. TUG has a network of "site coordinators" who serve as focal points of communication for people with the same computer configurations. Occasional short courses are given in order to provide concentrated training in special topics; videotapes of these courses are available for rental. Meetings of the entire TUG membership are held at least once a year. You can buy TEX T-shirts at these meetings. Information about membership in TUG and subscription to *TUGboat* is available from

TEX Users Group
Email: `TUG@tug.org`
Internet: `http://www.tug.org/`

TEX in Russia

TEX in Russia made its appearance in the mid-1980's. At first, TEX was only known to the scientists who traveled to scientific institutes in Western Europe and the USA. They were, as a rule, the physicists, including scientists from the Institute of High Energy Physics (IHEP, in Protvino). The first Cyrillic version of TEX appeared at that very institute. It became known as "Protvinskaya"; the authors were S. Klimenko, B. Malyshev, A. Samarin, et al. The first Cyrillic font in this version was `tt`: according to the rules of thesis preparation at the time, they had to be typed on a typewriter (not a computer!). The young scientists devised a way to deceive the bureaucrats, so that they did not have to retype materials that were prepared on computers.

1st Cyr. version

tt font

By the end of the 80's and the beginning of the 90's, TEX had lost its exotic nature. There were some specialists and groups of scientists who began using TEX with some Cyrillic fonts and to implement their own Cyrillic versions of TEX.

AMS training

In an agreement between the American Mathematical Society and Soviet publishing houses (Mir and Nauka) and Leningrad State University, three Russian specialists were sent to the AMS for $\mathcal{AMS}$-TEX training. During this visit, they called on the TUG office and the idea to create *CyrTUG* in the USSR was born.

CyrTUG

In the spring of 1991 (May 23–24), there was a "constituent assembly" of *CyrTUG* at Mir Publishers. There were 23 TEX users from Moscow, Protvino, St. Petersburg and Novosibirsk at the meeting. The participants reported on their work on Cyrillic versions of TEX, and the President, Executive Director, and the Board were elected. The Cyrillic TEX Users Group (or, in Russian: Associaciĭa Polzovateleĭ Kirillicheskogo TEX'a) was born.

CyrTUG memb.

For the five years *CyrTUG* has been in existence, some 700 TEX users have been members, with about 50 scientific institutes, universities, and publishing houses as institutional members. There have also been citizens of the USA, Slovak Republic, Czech Republic, Switzerland, and the Netherlands; institutional members include CERN (Geneva), JINR (Dubna), Mir Publishers (Moscow), MekhMat faculty of Moscow State University, UrbanSoft (St. Petersburg), Institute of Mathematics (Kazan), Electrotechnical Institute (Novosibirsk), Institute of Mathematics and Mechanics (Ekaterinburg), and others. The Grand Wizard, Donald Knuth, is an honorary member of *CyrTUG*: during his visit to St. Petersburg he was presented with card No. 0314.

LATEX

Among the various macro packages developed for TEX, LATEX probably is the most widely used.

history

LATEX was originally written by Leslie Lamport. In 1994 the LATEX package was updated by the LATEX3 team, led by Frank Mittelbach, to include some long-requested improvements, and to reunify all the patched versions which had cropped up since the release of LATEX 2.09 some years earlier. To distinguish the new version from the old, it is called LATEX 2_ε.

LATEX is much easier and safer to work with than TEX; it has a number of built-in safety features and a large set of error messages.

add. features

LATEX, building on TEX, provides the following additional features:

logical units

- An article is divided into *logical units* such as an abstract, sections, theorems, a bibliography, and so on. The logical units are typed separately. After all the units have been typed, LATEX organizes the *placement* and *formatting* of these elements.

bookkeep. chores

- LATEX relieves you of tedious *bookkeeping chores*. Consider a completed article, with theorems and equations numbered and properly cross-referenced. Upon final reading, some changes must be made, for example, section 4 has to be placed after section 7, and a new theorem has to be inserted somewhere in the middle. Such a minor change used to be a major headache! But with LATEX, it becomes almost a pleasure to make such changes. LATEX automatically redoes all the numbering and cross-references.

bibl. reference

- Typing the same *bibliographic references* in article after article is a tedious chore. With LaTeX you may use BibTeX, a program that helps you create and maintain bibliographic databases, so references need not be retyped for each article. BibTeX will select and format the needed references from the databases.

1.5. METAFONT and METAPOST

METAFONT

METAFONT was developed by Donald Knuth as part of the TeX typesetting system. METAFONT is a graphics programming language (like PostScript) designed for producing complete typeface families, but it has applications wider than just fonts—it can also produce geometric designs, dingbats, etc. And it has considerable mathematical and equation-solving capabilities which can be useful entirely on their own.

batch lang.

METAFONT is a batch language, like C or Pascal: you compile a METAFONT program into a corresponding font, rather than interactively drawing lines or curves. This approach

disadvantage

has both considerable disadvantages (people unfamiliar with conventional programming

advantage

languages will be unlikely to find it usable) and considerable advantages (you can make your design intentions specific and parameterizable).

metafonts

Metafonts exhibit some very desirable qualities. One of the important features is

scaling

that metafonts can scale very gracefully. The metafont Computer Modern has different shape at 20 point and 10 point. The shape changes with size, because it is desirable for a smaller font to be proportionately wider than a larger font (this makes the larger fonts more elegant and the smaller fonts more readable). Prior to Adobe's multiple master technology, METAFONT was unique with respect to having this feature.

not ubiquitous

It's main weakness is that it is not as ubiquitous as TrueType or Type 1. It is also not

WYSIWYG

quite suited to WYSIWYG publishing. Of course, this isn't a major disadvantage when TeX is your typesetting system.

METAPOST

METAPOST is a programming language much like Knuth's METAFONT except that it outputs PostScript programs instead of bitmaps. It was developed by John D. Hobby based on the sources for METAFONT.

borrowed tools

Borrowed from METAFONT are the basic tools for creating and manipulating pictures. These include numbers, coordinate pairs, cubic splines, affine transformations, text strings,

add. features

and boolean quantities. Additional features facilitate integrating text and graphics and accessing special features of PostScript such as clipping, shading, and dashed lines. Another feature borrowed from METAFONT is the ability to solve linear equations that are given implicitly, thus allowing many programs to be written in a largely declarative style. By building complex operations from simpler ones, METAPOST achieves both power and flexibility.

purpose

METAPOST is particularly well-suited to generating figures for technical documents where some aspects of a picture may be controlled by mathematical or geometrical constraints that are best expressed symbolically. In other words, METAPOST is not meant to take the place of a freehand drawing tool or even an interactive graphics editor. It is really a programming language for generating graphics, especially figures for TEX and troff documents. The figures can be integrated into a TEX document via a freely available program called dvips as shown in Fig. 1. A similar procedure works with troff: the dpost output processor includes PostScript figures when they are requested via troff's \X command.

figs. in TEX

figs. in troff

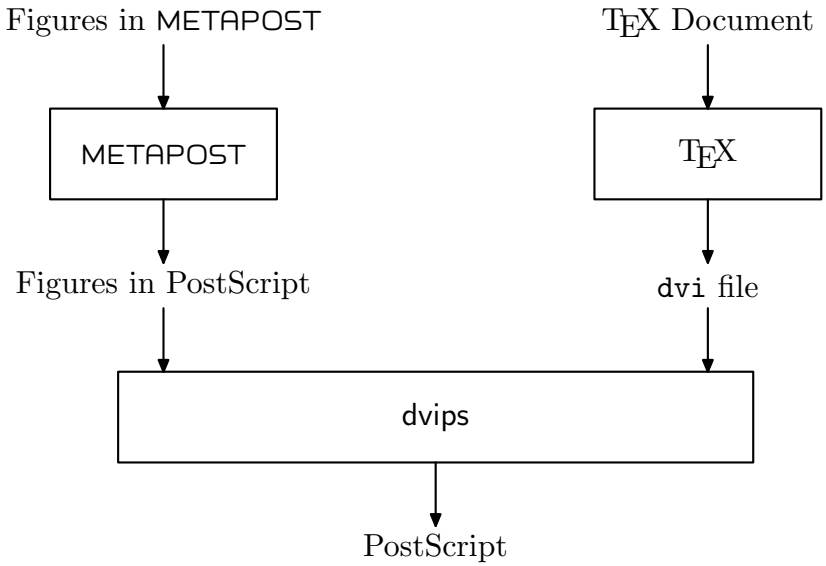

Fig. 1. A diagram of the processing for a TEX document with figures in METAPOST

integr. text

METAPOST is able to easily include TEX or LATEX text within its graphic output, so tags on graphs and drawings can match the text font exactly. The METAPOST package includes special macro packages for drawing graphs and for drawing boxes and ovals. See Fig. 2 for the examples of what can be achieved with METAPOST.

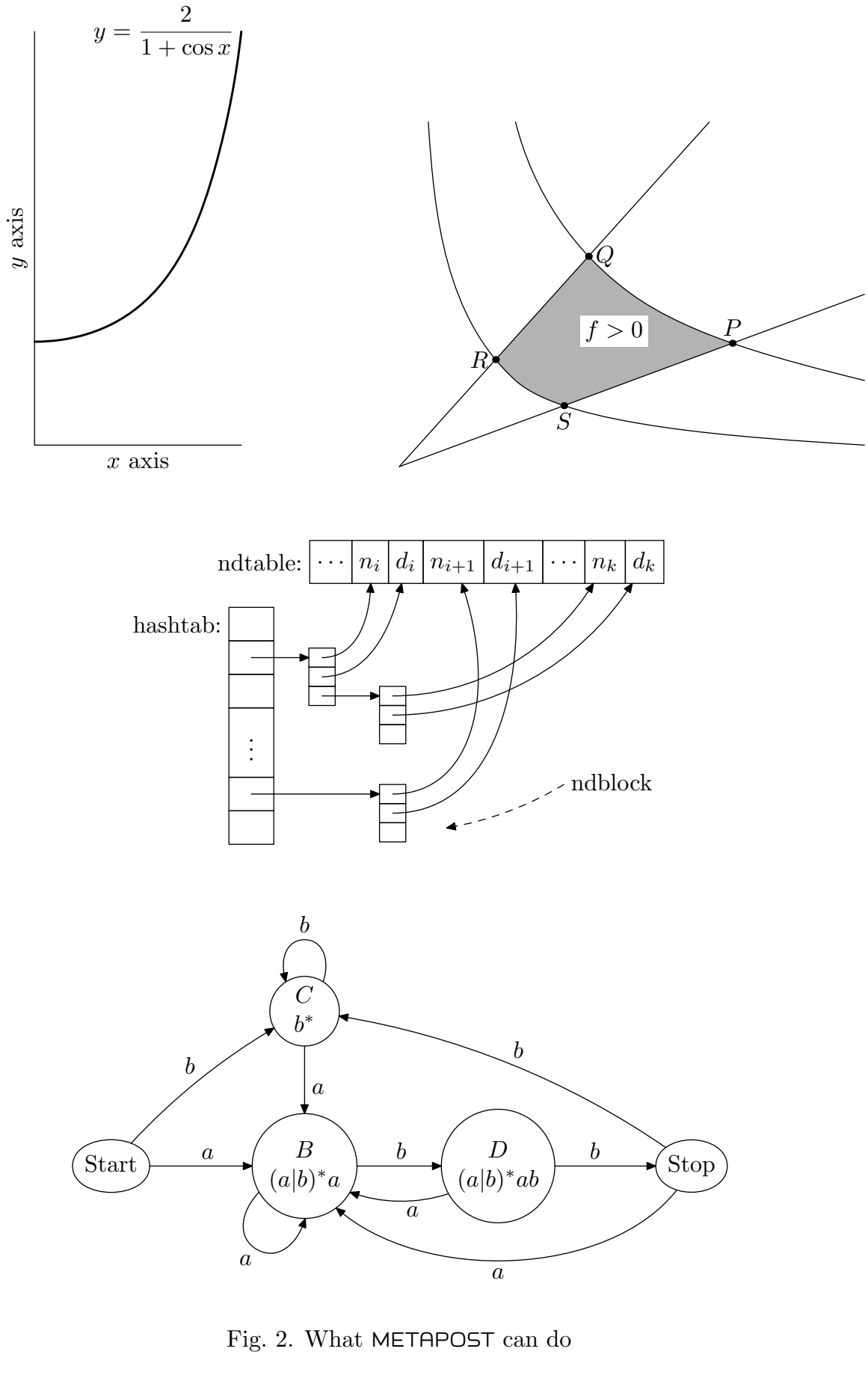

Fig. 2. What METAPOST can do

Part 2
RICHARD M. STALLMAN

2.0. Background

Richard M. Stallman is the initiator and the main ideologist of the Free Software Foundation, an organization founded to promote free software development. He authored such software as GNU Emacs, gcc, gdb, and GNU make. The free software movement sparked by Stallman had and will continue to have many significant consequences not only for the computer world but for the entire society.

2.1. Biography

education

Stallman graduated from Harvard in 1974 with a B.A. in physics. During his college years, he also worked as a staff hacker at the MIT Artificial Intelligence Lab, learning operating system development by doing it. He wrote the first extensible Emacs text editor there in 1975. In January 1984 he resigned from MIT to start the GNU project.

awards

Stallman received the Grace Hopper award in 1991 from the Association for Computing Machinery, for his development of the first Emacs editor. In 1990 he was awarded a MacArthur foundation fellowship, and in 1996 an honorary doctorate from the royal institute of Technology in Sweden. In 1998 he received the Electronic Frontier Foundation's pioneer award along with Linus Torvalds. In 1999 he received the Yuri Rubinski award. In 2001 he received a second honorary doctorate, from the University of Glasgow, and shared the Takeda award for social/economic betterment with Torvalds and Ken Sakamura. In 2002 he was elected to the National Academy of Engineering.

2.2. The GNU Project

The first software-sharing community

When Richard Stallman started working at the MIT Artificial Intelligence Lab in 1971, he became part of a software-sharing community that had existed for many years. Sharing of software was not limited to that particular community; it is as old as computers, just as sharing of recipes is as old as cooking.

ITS OS

The AI Lab used a timesharing operating system called ITS (the Incompatible Timesharing System) that the lab's staff hackers had designed and written in assembler language for the Digital PDP–10, one of the large computers of the era. As a member of this community, an AI lab staff system hacker†, Stallman's job was to improve this system.

† The use of "hacker" to mean "security breaker" is a confusion on the part of the mass media. Hackers refuse to recognize that meaning, and continue using the word to mean, "Someone who loves to program and enjoys being clever about it."

orig. free sware The community did not call their software "free software," because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, they gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that *mutual benefit* you could read it, change it, or cannibalize parts of it to make a new program. The people who provided the source code gained in the process, as the borrower would have introduced his/her own additional features to the program, features that everybody was perfectly free to borrow in return.

The collapse of the community

PDP discont.'d The situation changed drastically in the early 1980's when Digital discontinued the PDP–10 series. Its architecture, elegant and powerful in the 60's, could not extend naturally to the larger address spaces that were becoming feasible in the 80's. This meant that nearly all of the programs composing ITS were obsolete.

hired away The AI lab hacker community had already collapsed, not long before. In 1981, the spin-off company Symbolics had hired away nearly all of the hackers from the AI lab, and *Digital's* the depopulated community was unable to maintain itself. When the AI lab bought a new PDP–10 in 1982, its administrators decided to use Digital's non-free timesharing system instead of ITS.

other non-free The modern computers of the era, such as the VAX or the 68020, had their own operating systems, but none of them were free software: you had to sign a nondisclosure agreement even to get an executable copy.

The infamous Xerox printer

At this time, Richard Stallman had an experience which considerably influenced his subsequent life and would lead ultimately to the inception of the GNU Project.

Xerox donation Xerox Corporation had donated a new laser printer to AI lab, a common practice with big companies. A cutting edge prototype, it was a modified version of the popular Xerox *flaws* photocopier. But it wasn't until a few weeks after its arrival that the machine's flaws began to surface. Chief among the drawbacks was the machine's inherent susceptibility to *paper jams* paper jams. It was quite frustrating, as users had to run back and forth to check that the printer was not jammed once again.

similar problem Years before, when the lab was still using its old printer, Stallman had solved a similar problem by opening up the printer driver on the lab's PDP–11 machine. Stallman couldn't eliminate paper jams, but he could add code that ordered the PDP–11 to check the printer periodically and report back to the PDP–10, the lab's central computer. To ensure that one user's negligence didn't bog down an entire line of print jobs, Stallman also added code that instructed the PDP–10 to notify every user with a waiting print job that the printer was jammed. Because the message went out to the people with the most pressing need to fix the problem, chances were higher that the problem got fixed in due time. A minute or two after the printer got in trouble, the two or three people who got messages would arrive to fix the machine. Of those two or three people, one of them, at least, would usually know how to fix the problem.

update "hack"

When Stallman spotted the print-jam defect in the new Xerox laser printer, he didn't panic. He simply looked for a way to update the old "hack" for the new system. In the course of looking up the Xerox laser-printer software, however, Stallman made a troubling discovery. The printer didn't have any software, at least nothing Stallman or a fellow programmer could read. Until then, most companies had made it a form of courtesy to publish source-code files for all software. Xerox, in this instance, had provided software files in binary form. But the notion of information sharing was so central to the hacker culture that Stallman knew it was only a matter of time before some hacker in some university lab or corporate computer room proffered a version of the laser-printer software source code.

no sources

matter of time

rumor

In a short while, Stallman learned that a scientist at the computer-science department at Carnegie Mellon University had just departed a job at the Xerox Palo Alto Research Center. Not only had the scientist worked on the laser printer in question, but according to rumor, he was still working on it as part of his research duties at Carnegie Mellon.

visit Carnegie M.

Within a few months, Stallman had a business-related reason to visit the Carnegie Mellon campus. During that visit, he made sure to stop by the computer-science department. Department employees directed him to the office of the faculty member leading the Xerox project. When Stallman reached the office, he found the professor working there.

professs. refused

nondiscl. agr.

After briefly introducing himself as a visitor from MIT, Stallman requested a copy of the laser-printer source code so that he could port it to the PDP–11. To his surprise, the professor refused to grant his request. The professor's unwillingness to hand over the source code stemmed from a nondisclosure agreement, a contractual agreement between the professor and the Xerox Corporation giving the professor access to the software source code in exchange for a promise of secrecy. Now a standard item of business in the software industry, the nondisclosure agreement, or NDA, was a novel development at the time, a reflection of both the commercial value of the laser printer to Xerox and the information needed to run it.

RMS's percept.

For Stallman, however, the NDA was something else entirely. It was a refusal on the part of Xerox to participate in a system that, until then, had encouraged software programmers to regard programs as communal resources. Stallman's attempt to drop in on a fellow programmer unannounced had been intended as a demonstration of neighborliness.

quote

Now that the request had been refused, it felt like a major blunder. "I was so angry I couldn't think of a way to express it. So I just turned away and walked out without another word," Stallman recalls. "I might have slammed the door. Who knows? All I remember is wanting to get out of there."

A stark moral choice

With the community gone, to continue as before was impossible. Instead, Stallman faced a stark moral choice.

easy choice

another choice

The easy choice was to join the proprietary software world, signing nondisclosure agreements. Another choice, straightforward but unpleasant, was to leave the computer field. None of these choices seemed to please Stallman, so he asked himself, was there

a program or programs that he could write, so as to make the community possible once again?

OS needed 1st

What was needed first was an operating system. That is the crucial software for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. A free operating system could make a community of cooperating hackers possible.

right skills

UNIX compat.

As an operating system developer, Stallman had the right skills for this job. He chose to make the system compatible with UNIX because the overall design was already proved and portable, and because compatibility made it easy for UNIX users to switch from UNIX to GNU. The name GNU was chosen following a hacker tradition, as a recursive acronym for "GNU's Not UNIX."

name "GNU"

not just a kernel

An operating system does not mean just a kernel, barely enough to run other programs. In the 1970's, every operating system worthy of the name included command processors, assemblers, compilers, interpreters, debuggers, text editors, mailers, and much more. ITS had them, Multics had them, VMS had them, and UNIX had them. The GNU operating system would include them too.

GNU software and the GNU system

uses existing sw.

Developing a whole system is a very large project. To bring it into reach, Stallman decided to adapt and use existing pieces of free software wherever that was possible. For example, he decided at the very beginning to use TEX as the principal text formatter; a few years later, he decided to use the X Window System rather than writing another window system for GNU.

Because of this decision, the GNU system is not the same as the collection of all GNU software. The GNU system includes programs that are not GNU software, programs that were developed by other people and projects for their own purposes, but which could be used because they are free software.

Commencing the project

quits MIT

why quit

In January 1984 Stallman quit his job at MIT and began writing GNU software. Leaving MIT was necessary so that MIT would not be able to interfere with distributing GNU as free software. If Stallman had remained on the staff, MIT could have claimed to own the work, and could have imposed their own distribution terms, or even turned the work into a proprietary software package.

The first steps

VUCK

Shortly before beginning the GNU project, Stallman heard about the Free University Compiler Kit, also known as VUCK. (The Dutch word for "free" is written with a V.) This was a compiler designed to handle multiple languages, including C and Pascal, and to support multiple target machines. Stallman wrote to its author asking if GNU could use it.

He responded derisively, stating that the university was free but the compiler was not. Stallman therefore decided that his first program for the GNU Project would be a multi-language, multi-platform compiler.

Pastel

Hoping to avoid the need to write the whole compiler himself, Stallman obtained the source code for the Pastel compiler, which was a multi-platform compiler developed at Lawrence Livermore Lab. It supported, and was written in, an extended version of Pascal, designed to be a system-programming language. Stallman added a C front end, and began porting it to the Motorola 68000 computer. But Stallman had to give that up when he discovered that the compiler needed many megabytes of stack space, and the available 68000 UNIX system would only allow 64k.

GCC

At this point, Stallman concluded he would have to write a new compiler from scratch. That new compiler is now known as GCC; none of the Pastel compiler is used in it. But that was some years later; first, Stallman worked on GNU Emacs.

GNU Emacs

Emacs is an advanced, extensible, customizable, self-documenting real-time display editor. Stallman began work on GNU Emacs in September 1984, and in early 1985 it was beginning to be usable.

distr. via ftp
no job
selling tapes
FS dist. business

At this point, people began wanting to use GNU Emacs, which raised the question of how to distribute it. Stallman put it on the anonymous ftp server on the MIT computer that he used. But at that time, many of the interested people were not on the Internet and could not get a copy by ftp. Stallman had no job, and he was looking for ways to make money from free software. So he announced that he would mail a tape to whoever wanted one, for a fee of $150. In this way, Stallman started a free software distribution business, the precursor of the companies that today distribute entire Linux-based GNU systems.

Free as in freedom

The term "free software" is sometimes misunderstood—it has nothing to do with price. It is about freedom. Here, therefore, is the definition of free software—a program is free software, for you, a particular user, if:

- You have the freedom to run the program, for any purpose.
- You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)
- You have the freedom to redistribute copies, either gratis or for a fee.
- You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.

no contradiction
raise funds

Since "free" refers to freedom, not to price, there is no contradiction between selling copies and free software. In fact, the freedom to sell copies is crucial: collections of free software sold on CD-ROMs are important for the community, and selling them is an important way to raise funds for free software development.

altern. for "free" Because of the ambiguity of "free," people have long looked for alternatives, but no one has found a suitable alternative. The English Language has more words and nuances than any other, but it lacks a simple, unambiguous word that means "free," as in freedom—"unfettered" being the word that comes closest in meaning. Such alternatives as "liberated," "freedom," and "open" have either the wrong meaning or some other disadvantage.

Copyleft and the GNU GPL

can go propriet. If a program is free software when it leaves the hands of its author, this does not necessarily mean it will be free software for everyone who has a copy of it. For example, public domain software (software that is not copyrighted) is free software; but anyone can make a proprietary modified version of it. Likewise, many free programs are copyrighted but distributed under simple permissive licenses which allow proprietary modified versions.

protect freedom The goal of GNU was to give users freedom. So the GNU Project needed to use distribution terms that would prevent GNU software from being turned into proprietary software. The method they use is called "copyleft."

flips copyright Copyleft uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of a means of privatizing software, it becomes a means of keeping software free.

can't restrict The central idea of copyleft is that everyone has permission to run the program, copy the program, modify the program, and distribute modified versions—but not permission to add restrictions of their own. Thus, the crucial freedoms that define "free software" are guaranteed to everyone who has a copy; they become inalienable rights.

modif.'s also free For an effective copyleft, modified versions must also be free. This ensures that work based on the GNU Project's work becomes available to their community if it is published. When programmers who have jobs as programmers volunteer to improve GNU software, it is copyleft that prevents their employers from saying, "You can't share those changes, because we are going to use them to make our proprietary version of the program."

combine f./non-f. A related issue concerns combining a free program with non-free code. Such a combination would inevitably be non-free; whichever freedoms are lacking for the non-free part would be lacking for the whole as well. So anything added to or combined with a copylefted program must be such that the larger combined version is also free and copylefted.

other copylefts The specific implementation of copyleft that the GNU Project uses for most GNU software is the GNU General Public License, or GNU GPL for short. There are other kinds of copyleft that are used in specific circumstances. GNU manuals are copylefted also, but use a much simpler kind of copyleft, because the complexity of the GNU GPL is not necessary for manuals.

The Free Software Foundation

As interest in using Emacs was growing, other people became involved in the GNU project, and the GNU Project decided that it was time to seek funding once again. So in 1985 they created the Free Software Foundation, a tax-exempt charity for free software development. The FSF also took over the Emacs tape distribution business; later it extended this by

adding other free software (both GNU and non-GNU) to the tape, and by selling free manuals as well.

income from sales The FSF accepts donations, but most of its income has always come from sales—of copies of free software, and of other related services. Today it sells CD-ROMs of source code, CD-ROMs with binaries, nicely printed manuals, all with freedom to redistribute and modify, and Deluxe Distributions (where they build the whole collection of software for your choice of platform).

writes sware Free Software Foundation employees have written and maintained a number of GNU
GNU libc software packages. Two notable ones are the C library and the shell. The GNU C library is what every program running on a GNU/Linux system uses to communicate with Linux. It was developed by a member of the Free Software Foundation staff, Roland McGrath.
BASH The shell used on most GNU/Linux systems is BASH, the Bourne Again Shell, which was developed by FSF employee Brian Fox.

The GNU Hurd, Linux and GNU/Linux

lacking kernel By 1990, the GNU system was almost complete; the only major missing component was the kernel. The GNU Project had decided to implement their kernel as a collection of
Mach server processes running on top of Mach. Mach is a microkernel developed at Carnegie
GNU HURD Mellon University and then at the University of Utah; the GNU HURD is a collection
start delayed of servers that run on top of Mach, and do the various jobs of the UNIX kernel. The start of development was delayed as the GNU Project waited for Mach to be released as free software, as had been promised. Furthermore, making the HURD work solidly has stretched on for many years.

Linux is ready So the GNU Hurd is not yet ready for production use. Fortunately, another kernel is available. In 1991, Linus Torvalds developed a UNIX-compatible kernel and called it
free OS complete Linux. Around 1992, combining Linux with the not-quite-complete GNU system resulted in a complete free operating system. (Combining them was a substantial job in itself, of course.) It is due to Linux that we can actually run a version of the GNU system today.

"GNU/Linux" The GNU community calls this system version GNU/Linux, to express its composition as a combination of the GNU system with Linux as the kernel.

Part 3
LINUS TORVALDS

3.0. Background

Linus Torvalds is the creator of the Linux kernel, used in the many GNU/Linux operating system variants. By now, thousands of programmers have contributed to the development of the kernel, yet Torvalds was the creator, and the mastermind behind the original. Torvalds made a huge impact on emerging developers, and an even larger one on the computer users community in general.

3.1. Biography

Linus Torvalds was born in Helsinki, Finland on December 28, 1969 to Nils and Mikke
non-comp. envir. Torvalds. Surprisingly, for most of his life Linus Torvalds lived in a very non-computer oriented environment. Both his parents were very outspoken liberalists and journalists by
Commodore trade. Linus' first exposure to computers was around 1980 when his grandfather bought a Commodore VIC–20, which he used to analyze statistical data. He greatly encouraged Linus to use the new machine and Linus listened to his grandfather's encouragement. His
1st computer passion for computers and programming grew exponentially. His first computer was a Sinclair QL, which was a common PC alternative in Europe at the time. He instantly took to developing and on his own learned BASIC, Assembly, and then C—now a common development practice among programming curricula.

education In the fall of 1988, Torvalds enrolled in Helsinki University as a Computer Science major. There he honed his skills as a programmer and was exposed to larger, faster, and
started Linux more common systems. By 1991, he had enough money to buy his own 386sx PC. He then started writing the piece of software that would make his name known worldwide.

awards In 1997 Linus Torvalds received 1997 Nokia Foundation Award. In March of 1997 he received Lifetime Achievement Award at Uniforum Pictures. In 1998 Torvalds received the Electronic Frontier Foundation's pioneer award along with Richard Stallman. In 2001 he shared the Takeda award for social/economic betterment with Stallman and Ken Sakamura.

3.2. Linux kernel. GNU/Linux

First usable kernel

Torvalds started with basic x86 assembly language, which all Intel-based computers used, and soon became quite comfortable with the new system. However, he was not satisfied
DOS & XENIX with the software that was available for it. MS-DOS was available for the average user, and for a while XENIX was available for the high-end users. However, even XENIX

Minix seemed incomplete to him. Minix, another current operating system developed by Andy Tannenbaum, was another possibility, but because of its initial success was licensed and copyrighted by Prentice Hall and was soon corrupted with corporate marketing ploys.

"Linus's Minix" Linus decided to create his own PC-based version of UNIX, and in 1991 Linus started coding "Linus' Minix," or "Linux."

Months of determined programming work yielded the beginnings of an operating system kernel known as Linux that, eight years later, developed into what many observers saw as a genuine threat to mighty Microsoft and its seemingly ubiquitous Windows OS.

ran bash, gcc, ... The kernel has finally reached the stage where it was usable for some purposes—it *posted message* could successfully run bash, gcc, GNU make, GNU sed, compress. Torvalds posted a message on the Internet to alert other PC users to his new system. He made the software available for free downloading and released the source code, which meant that anyone with knowledge of computer programming could modify Linux to suit their own purposes. Linux soon had a following of enthusiastic supporters who, because they had access to the source code, were able to help Torvalds retool and refine the software.

Combining the GNU Project's operating system with Linux

add. software But the kernel was not yet an operating system—to be of any use to its users, it had to be augmented by additional software, including both system software (command language interpreters, compilers, libraries, back-up software, telnet, etc.) and so-called "application software" (graphical desktop environments, spreadsheets, games, graphics viewers, video players, etc.).

make themselves use proprietary Torvalds and his followers could choose to develop all that software from scratch, but this was a daunting task which would take years to accomplish. Of course, there was the alternative of using some other, proprietary software, but that would defy the idea behind the Linux kernel to become "free" software which anybody could use and modify for the

GNU Project benefit of the entire community. Fortunately, the GNU Project (initiated by Richard M. Stallman) was already working on such free operating system for almost a decade. The GNU system was lacking this very component—the kernel. Around 1992, combining Linux with the not-quite-complete GNU system resulted in a complete free operating system.

"GNU/Linux" Combining them was a substantial job in itself, of course. The resulting system is commonly referred to as "GNU/Linux," to express its composition as a combination of the GNU system with Linux as the kernel.

Collaborative development. FreeBSD project

Minix contrib. Communicating heavily with other Minix coders over USENET groups, Linus' kernel soon took shape. Without the help of his Minix community counterparts, Linus could not have written his kernel, and it would not have become what it is today.

Other factors also helped the growth of the Linux kernel. In the early 1990's, mostly 92–94, the GNU/Linux operating system was shaping up to be a full-featured developer-

FreeBSD centered operating system. However, at the same time another similar project was taking off in the UNIX community, one controlled and developed by a whole team of experts.

This project was FreeBSD, also a port of the UNIX operating system for the x86 processor architecture. For a time it seemed that the two operating systems were in a heated competition to release the best and biggest before the other. Though, in retrospect, this occurrence fueled the speedy development of both operating systems. Both thrived in their own right. In the end, a nasty lawsuit from the UNIX System Laboratories (a subsidiary of AT&T), brought the development of FreeBSD to a temporary halt, and indirectly provided great marketing value for the Linux project.

nasty lawsuit

After Linus released early versions of his kernel as free software, developers were constantly helping the development of new features. Though it would seem that an unknown group of hundreds (now thousands) of developers all working on the same code all at once would end up in a mess of unusable code, it turned out to be quite the opposite. The product of this system was a quickly developed, fast, stable, secure operating system for developers, by developers.

seeming mess

Rising popularity. Support by commercial companies

Operating GNU/Linux required a certain amount of technical acumen; it was not as easy to use as more popular operating systems such as Windows, Apple Computer Inc.'s Mac OS, or IBM's OS/2. Because its volunteer developers prided themselves on the quality of their work, however, GNU/Linux evolved into a remarkably reliable, efficient system that rarely crashed. GNU/Linux got its big break in the late 1990's when competitors of Microsoft began taking the upstart OS seriously. Netscape Communications Corp., Corel Corp., Oracle Corp., Intel Corp., and other companies announced plans to support GNU/Linux as an inexpensive alternative to Windows.

By 1999 Torvalds had become a cult hero to a devoted band of computer users. An estimated seven million computers were running on GNU/Linux by 1999, still available free of charge. Meanwhile, Torvalds had taken a position with Transmeta Corp., owned by Microsoft co-founder Paul Allen.

Transmeta Corp.

Linux Distributions

After GNU/Linux gained "release" quality, meaning it was stable enough to be used on a day-to-day basis under any circumstance, Linus' role in development of the kernel seemed to fade away. Because of the overwhelming popularity of the project and the GNU license on the code, people started packaging their own fully functional operating systems and distributed them to a niche of users. These operating systems were just preconfigured GNU/Linux systems. This became the basis of what is now commonly called a "distro," meaning GNU/Linux distribution. Linus' main role then was just a Linux kernel distributor. Although he still made some administrative decisions pertaining to the development, he did not do much coding at all.

kernel distr.'tor

not much coding

Conclusion

The GNU/Linux operating system now enjoys thousands of developers constantly working on the code, and millions others developing software to be run on it, not to mention the millions of end-users running GNU/Linux, or some Linux distribution. Because of one

man's ambition, the computing world has never been the same. Millions of people have enjoyed helping development of the operating system and millions of others have enjoyed using the quality operating system as an alternative to other inferior products.

Richard M. Stallman, Linus Torvalds, and Donald E. Knuth engage in a discussion on whose impact on the computerized world was the greatest. Stallman: "God told me I have programmed the best editor in the world!" Torvalds: "Well, God told me *that I have programmed the best operating system in the world!" Knuth: "Wait, wait—I never said that."*

— *From* `news://rec.humor.funny`
submitted by `ermel@gmx.de` (ERIK MELTZER)

Bibliography

I have not given formal references in the text as they are hardly appropriate in a document of this nature. Following is the list of publications pertaining to the above discussion and/or used in the preparation of this document, which the reader may find interesting.

Donald E. Knuth. *The Art of Computer Programming.* Addison–Wesley, 1998, third edition, ISBN 0–201–89683–4.

Ronald L. Graham, Donald E. Knuth, Oren Patashnik. *Concrete Mathematics: a Foundation for Computer Science.* Addison–Wesley, 1994, second edition, ISBN 0–201–55802–5.

Donald E. Knuth. *Mathematical Typography. Bulletin of the American Mathematical Society* (new series) 1 (March 1979), 337–372. [Reprinted as part 1 of *TEX and METAFONT: New Directions in Typesetting* (Providence, R.I: American Mathematical Society, and Bedford, Mass: Digital Press, 1979).]

Donald E. Knuth. *Tau Epsilon Chi, a system for technical text.* Stanford Computer Science Report 675 (Stanford, California, September 1978), 198 pp. [Reprinted as part 2 of *TEX and METAFONT*, the book cited above.]

Donald E. Knuth. *The* `WEB` *system of structured documentation.* Stanford Computer Science Report 980 (Stanford, California, September 1983), 206 pp.

Donald E. Knuth. *Literate programming. The Computer Journal* 27 (1984), 97–111.

Literate programming site. `http://www.literateprogramming.com/`

Donald E. Knuth. *The TEXbook.* Addison–Wesley, 1984.

Norman Walsh. *Making TEX Work.* Available on-line: `http://MakingTeXWork.SourceForge.net/mtw/index.html`

TUG (TEX Users Group). Email: `TUG@tug.org`; Internet: `http://www.tug.org/`

TUGboat. `ftp://tug.org/`

Philip Taylor. *Computer Typesetting or Electronic Publishing? New trends in scientific publication. TUGboat,* 17(4) (1996), 367–381.

Irina A. Makhovaya. *TEX in Russia: ab ovo or About the TEXnical evolution in Russia. TUGboat,* 17(3) (1996), 259–264.

Jeffrey McArthur. *Developing Database Publishing Systems Using TEX. TUGboat,* 19(2) (1998), 188–194.

Alan Hoenig. *Introducing METAPOST. TUGboat,* 16(1) (1995), 45.

D. P. Story. *AcroTEX: Acrobat and TEX Team Up. TUGboat,* 20(3) (1999), Proceedings of the 1999 Annual Meeting, 196–201.

Bibliography

Bart Childs. *Teaching CS/1 Courses in a Literate Manner. TUGboat,* 16(3) (1995), Proceedings of the 1995 Annual Meeting, 300–309.

Leslie Lamport. *LaTeX: A Document Preparation System.* Addison–Wesley, Reading, Massachusetts, second edition, 1994, ISBN 0–201–52983–1.

George A. Grätzer. *Math into LaTeX: an introduction to LaTeX and AMS-LaTeX.* Birkhäuser Boston, 1996.

Tobias Oetiker, Hubert Partl, Irene Hyna, Elisabeth Schlegl. *The Not So Short Introduction to LaTeX 2ε.* `CTAN:info/lshort/english`

AcroTeX's home page. `http://www.math.uakron.edu/~dpstory/acrotex.html`

Donald E. Knuth. *The METAFONTbook.* Addison–Wesley, Reading, Massachusetts, 1986. Volume C of *Computers and Typesetting.*

John D. Hobby. *A User's Manual for METAPOST.* AT&T Bell Laboratories.

GNU's home page. `http://www.gnu.org/home.html`

Richard Stallman's personal home page. `http://www.stallman.org`

Sam Williams. *Free as in Freedom: Richard Stallman's Crusade for Free Software.* O'Reilly & Associates, 2002. Also available on-line: `http://www.oreilly.com/openbook/freedom/index.html`

Linus Torvalds, David Diamond. *Just for Fun: the Story of an Accidental Revolutionary.* Harper Business, 2001.

Linux Magazine. `http://www.linux-mag.com/`

Linux Documentation Project. `http://sunsite.unc.edu/LDP/`

Lars Wirzenius, Joanna Oja. *The Linux System Administrators' Guide.* `http://www.iki.fi/viu/linux/sag/`

Eric S. Raymond. *The Cathedral and the Bazaar.* `http://www.tuxedo.org/~esr/writings/cathedral-bazaar`