Owing to the very high cost of mathematical printing it has been necessary to limit the alterations in this edition, in order that as much as possible of the earlier book may be reproduced by photographic processes. ... To save expense I have refrained from small alterations which would have been made, were I starting afresh, to improve the English, & c., being content to let what was printed stand, provided it was reasonably clear and not actually wrong.

Harold Hilton
*Plane Algebraic Curves,*
Oxford University Press, 1931

# TUGBOAT

THE TEX USERS GROUP NEWSLETTER
EDITOR ROBERT WELLAND

VOLUME 2, NUMBER 2 JULY 1981
PROVIDENCE RHODE ISLAND U.S.A.

## ADDRESSES OF OFFICERS, AUTHORS AND OTHERS

**BALL, George**
Computing Center
Washington State University
Pullman, WA 99164
509-335-6611

**BEEBE, Nelson**
Department of Physics
201 North Physics Building
University of Utah
Salt Lake City, UT 84112
801-581-5254

**BEETON, Barbara**
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

**DÍAZ, Max**
Mathematics Department
Stanford University
Stanford, CA 94305
415-327-8483

**DOHERTY, Barry**
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

**FRISCH, Michael J.**
University Computing Center
208 Union St. S.E.
University of Minnesota
Minneapolis, MI 55455
612-373-4599

**FUCHS, David**
Department of Computer Science
Stanford University
Stanford, CA 94305
415-497-1646

**GOUCHER, Raymond E.**
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

**HARRIS, Kent S.**
1026 West Maude, Suite 309
Sunnyvale, CA 94068
408-733-6617

**HICKEY, Thomas B.**
OCLC
6565 Frantz Road
Dublin, OH 43017
614-764-6000

**HODGE, Thea D.**
University Computing Center
208 Union St. S.E.
University of Minnesota
Minneapolis, MI 55455
612-373-4599

**HOWERTON, Charles P.**
National Bureau of Standards
Boulder Labs, room 3562
325 South Broadway
Boulder, CO 80303
303-499-1000 ext. 4433

**KELLER, Arthur M.**
Department of Computer Science
Stanford University
Stanford, CA 94305
415-497-3227

**KELLY, Bill**
Academic Computing Center
University of Wisconsin-Madison
1210 W. Dayton Street
Madison, WI 53706
608-262-8821

**KNUTH, Donald E.**
Department of Computer Science
Stanford University
Stanford, CA 94305

**LeVEQUE, William J.**
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

**LIANG, Frank**
Department of Computer Science
Stanford University
Stanford, CA 94305

**McCLURE, Robert S.**
1026 West Maude, Suite 309
Sunnyvale, CA 94068
408-733-6617

**McCOURT, Scott**
Burroughs Corporation
BCD Project
Corporate Drive, Commerce Park
Danbury, CT 06810
203-794-6191

**McKAY, Brendan D**
Vanderbilt University
Computer Science Department
Box 70, Station B
Nashville, TN 37235
615-322-6517

**MILLIGAN, Patrick**
Bell Northern Research, Inc.
685A Middlefield Road
Mountain View, CA 94043
415-969-9170, ext. 2837

**MORRIS, Robert**
Mathematics Department
UMASS at Boston
Boston, MA 02125
617-287-1900, ext. 2545

**NICHOLS, Monte C.**
Exploratory Chemistry Division
Sandia National Laboratory 8313
Livermore, CA 94550
415-422-2906

**PALAIS, Richard S.**
Department of Mathematics
Brandeis University
Waltham, MA 02154
(On leave through June 1982)

**PHILLIPS, Bob**
Oregon Software
2340 SW Canyon Road
Portland, OR 97201
503-226-7760

**PIERCE, Tom**
EG&G, Energy Measurements Group
P.O. Box 880
Collins Ferry Road
Morgantown, WV 26505
304-599-7585

**PIZER, Arnold**
Department of Mathematics
University of Rochester
Rochester, NY 14627
716-275-4428

**PLASS, Susan**
Polya 209
Center for Information Technology
Stanford University
Stanford, CA 94305
415-497-1322

**PRICE, Lynne A**
CALMA
Research and Development
212 Gibraltar Drive
Sunnyvale, CA 94086
408-744-1950

**SAMUEL, Arthur**
Computer Science Dept
Stanford University
Margaret Jacks Hall 436A
Stanford, CA 94305

**SCHWAB, Rachel L.**
National Institutes of Health
Computer Center Branch
Building 12B, Room 2N207
Bethesda, MD 20205

**SHERROD, Phil**
Box 1577, Station B
Vanderbilt University
Nashville, TN 37235
615-322-2951

**SMITH, Barry**
Oregon Software
2340 SW Canyon Road
Portland, OR 97201
503-226-7760

**SPIVAK, Michael**
2478 Woodridge Drive
Decatur, GA 30033
404-329-0372

**STOVALL, John**
7500 West Camp Wisdom Road
Dallas, TX 75236

**STROMQUIST, Ralph**
MACC
University of Wisconsin
1210 W. Dayton Street
Madison, WI 53706
608-262-8821

**TRABB-PARDO, Luis**
Department of Computer Science
Stanford University
Stanford, CA 94305

**WELLAND, Robert**
Department of Mathematics
Northwestern University
2033 Sheridan Road
Evanston, IL 60201
312-864-2898

**WHIDDEN, Samuel B**
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

**WHITNEY, Ronald**
American Mathematical Society
P O. Box 6248
Providence, RI 02940
401-272-9500

**ZABALA, Ignacio**
Department of Computer Science
Stanford University
Stanford, CA 94305

**ZAPF, Hermann**
Seitersweg 35
D-6100 Darmstadt Fed Rep Germany

**ZIPPEL, Richard**
Massachusetts Institute of Technology
545 Tech Square
Cambridge, MA 01239

**TUGboat is the newsletter of the TeX Users Group (TUG), and is published irregularly for TUG by the American Mathematical Society, P.O. Box 6248, Providence, RI 02940. Annual dues for individual members of TUG are $10.00 for 1981, and $15.00 for 1982; one subscription to TUGboat is included. Applications for membership in TUG should be addressed to the TeX Users Group, c/o American Mathematical Society, P.O. Box 1571, Annex Station, Providence, RI 02901; applications must be accompanied by payment.**

**Manuscripts should be submitted to the TUGboat Editor, Robert Welland, Department of Mathematics, Northwestern University, 2033 Sheridan Road, Evanston, IL 60201; submissions to the macros and problems columns should be sent to Lynne Price, CALMA, Research & Development, 212 Gibraltar Drive, Sunnyvale, CA 94086; items submitted on magnetic tape should be addressed to Barbara Beeton or Barry Doherty, American Mathematical Society, P.O. Box 6248, Providence, RI 02940.**

**Submissions to TUGboat are for the most part reproduced with minimal editing. Any questions regarding the content or accuracy of particular items should be directed to the authors.**

## OFFICIAL ANNOUNCEMENTS

### 1982 Membership Dues

1982 dues for individual members of TUG will be $15. Membership privileges will include all issues of TUGboat published during the membership (calendar) year. All new members and other persons inquiring about TUG will be sent TUGboat Vol. 1, No. 1, but 1981 issues will be sent only to persons paying the 1981 dues of $10. Beginning in 1982, foreign members will be able, on payment of a supplementary fee of $12 per subscription, to have TUGboat air mailed to them.

### TUGboat Schedule

Volumes of TUGboat are numbered on a calendar year basis. Volume 1 appeared in 1980, Volume 2 corresponds to 1981, and 1982 will bring Volume 3. Volume 1 consisted only of issue No. 1, dated October. Three issues are planned for Volume 2: No. 1 appeared in February, and No. 3 is planned for November. No schedule has been determined yet for 1982.

*The deadline for submitting items for Vol. 2, No. 3, is October 1, 1981. Contributions on magnetic tape or in manuscript form are encouraged; editorial addresses are given at the bottom of page 2, and a form containing instructions for submitting items on tape is bound into the back of this issue.*

It has been necessary to reprint back issues of TUGboat to fulfill the requirements of the growing membership. Each member is entitled to receive all issues which appear during his membership year, as well as Vol. 1, No. 1. If you have not received any issue to which you are entitled, instructions for obtaining such issues are included on the form referred to above.

* * * * * * * * * * *

## General Delivery

* * * * * * * * * * *

### EDITOR'S REMARKS

Robert Welland

We thank Lynne Price for taking on the responsibility of editing our Macro column; it is a complex task and we are thankful that it is in such talented hands. In the future, please submit all macros to

Lynne A. Price
CALMA
Research and Development
212 Gibralter Drive
Sunnyvale, CA 94086

We also thank Barry Smith of Oregon Software for getting TEX up and running on the VAX (see the VAX/VMS site report, page 34) and for making it easily available to all VAX users. Because of this work, we will see TEX flourish at very many sites.

Due to the hard work of Thea Hodge and Michael Frisch of the University of Minnesota (see their site report on page 28), we hope to see TEX up and running on Cyber machines sometime this fall; may the North Star guide them to success.

Lastly we extend the membership's gratitude to Barbara Beeton and Sam Whidden of the AMS whose hard work has made the TUGboat newsletters possible.

* * * * * * * * * * *

*Editor's note: The TUG Chairman, Richard Palais, is on leave for a year. At the Steering Committee meeting in May, Michael Spivak was appointed to serve as temporary Chairman until Dick's return.*

* * * * * * * * * * *

### CHAIRMAN'S REPORT

Michael Spivak

Since I am substituting for Dick Palais as Chairman of the TUG Steering Committee during the next year, I suppose that I ought to emerge briefly from the dimness of the AMS-TEX macro engine room and report on the view from the bridge.

Up here it's all inchoate brightness—everything's presently in a fog, though there's the promise of smooth sailing ahead. By the time of the Cincinnati meeting in January, the official Pascal TEX should be published, and more important, up and running at many more sites. If you have encountered and solved any particular problems bringing TEX up, your experiences will undoubtedly be of interest to others who want to implement TEX on the same, or similar, systems. If possible, please present your installation and/or use experiences at a session of the Cincinnati meeting; see the preliminary announcement by Tom Pierce on page 8. Perhaps we'll soon be able to stop worrying about getting TEX running, and can concentrate on using TEX. Two fundamentally opposed philosophies of how TEX should be supported were spelled out by Bob Morris and Sam Whidden in the last issue of TUGboat, and it will certainly be interesting to find out just *how* much support is going to be needed, since this will obviously influence the final decision. Actually, it seems that the problem of getting TEX running (i.e., producing dvi files) will be much easier to solve than the problem of getting the files printed, because of the variety of printers used and the secrecy about their inner workings. Perhaps this should be the next major problem that TUG could make a systematic attack on.

Of course, TEX is already up and being extensively used at some places, and more and more macro packages are being produced to get TEX to do just about everything except shine your shoes and write the papers for you. At the present stage, there are clearly still many tricks to be learned (as Don said, we are just beginning to scratch the tip of the iceberg). Even if a macro package performs some function that isn't of particular importance to another macro writer, it may contain some tricks that will be useful. Perhaps we should encourage more people to send in special tricks, or emphasize such tricks in their macro packages; eventually a "standard library" of tricks could be compiled. (Hours of pestering Don have produced some basic tricks, documented in the article "Macro Madness" (see page 50), that may help people to make TEX's macro facility work more like the 'programming language' that many have wished for.)

As this last paragraph has indicated, my own particular interest in using TEX is to get it to typeset anything a mathematician would want with minimal understanding on the part of the typist. Obviously the interests of other TEX users and implementors are going to be quite different. One of the problems with our last meeting was its undifferentiated nature. Although almost everyone got quite a bit out of some particular talk or meeting, it wasn't easy to know beforehand which one it would be. This is probably only to be expected at the initial stages, especially since so many different levels of TEXpertise are being addressed, but with Tom's help the Cincinnati meeting ought to be better structured, so that people can know what will be useful to them, and what can be skipped. Perhaps we'll even be so organized that we can propose the organization for the next meeting. Let's hope so!

* * * * * * * * * * *

## REPORT ON THE TUG STEERING COMMITTEE MEETING

Robert Morris

The TUG Steering Committee meeting took place in two sessions. The first, on May 13, simply set the loose agenda for the second, which was a public meeting on the evening of May 14.

The following actions were taken (a few of these may have been taken at the loosely organized general membership meeting on May 15):

a. By acclamation Mike Spivak was declared Chair of the Steering Committee. Richard Palais will be out of the country for a year.

b. Personal dues will be raised to $15 for 1982, but no institutional dues are contemplated pending TUG offering something to its members beyond the newsletter.

c. The Treasurer's report was approved; a version updated through June 30 appears on page 5. In summary, the individual membership fees and excess workshop revenue will cover the publication of TUGboat and minor administrative expenses for this year.

d. The idea of having architecture specific implementors' workshops, preferably at a successful site, was endorsed. These would be highly technical and financially self-supporting. Vanderbilt may organize one for TENEX sites; see page 28 in this issue for an announcement.

e. A tape standards committee was established to propose formats for the exchange of TEX files. A first proposal is put forth by Patrick Milligan on page 10.

f. Lynne Price agreed to edit the Macros and Problems columns in TUGboat, and to serve as focal point for discussion on the next generation of TeX, with emphasis on user-friendliness.

g. It was agreed to call a general membership meeting to coincide approximately with the winter meeting of the American Mathematical Society next January in Cincinnati; see the preliminary announcement by Tom Pierce on page 8.

h. Don Knuth announced his desire/intention to have TeX fully frozen by the end of the year, and to publish the theory and workings of TeX early in 1982.

i. It was decided that the architecture coordinators should not in general be those actually implementing, in order to shield the implementors from repetitive questions. This has worked well for the VAX/VMS implementation and will be gradually accomplished for the other architectures. Site coordinators are listed on the inside front cover, and their addresses are given on page 2. If you have/want current information, please contact them.

Minutes respectfully submitted,

Robert A. Morris
Secretary

*Editor's note: Attendees at the meeting may submit additions and corrections to the minutes in writing to the Secretary.*

* * * * * * * * * * *

## TUG TREASURER'S REPORT

June 30, 1981

| | | | |
|---|---|---|---|
| Beginning balance, January 1, 1981: | | | $( 419) |
| Income: Membership[1] | | $1,555 | |
| Tape leasing | | 400 | |
| Workshop[2] | | 7,445 | 9,400 |
| Expenses[3]: | | | |
| TUGboat Vol. 2, No. 1: 500 copies | | | |
| Printing | $1,012 | | |
| Postage | 320 | | |
| Clerical labor | 60 | $ 1,392 | |
| Reprinting TUGboat: | | | |
| Vol. 1, No. 1: 300 copies | | 195 | |
| Vol. 2, No. 1: 300 copies | | 655 | |
| Steering Committee luncheon, San Francisco Jan. 81 | | 170 | |
| Workshop[2] expenses | | 236 | ( 2,648) |
| Estimate of future 1981 income: | | | |
| Membership (100 members) | | $ 1,000 | |
| TeX tape sales/leasing | | 1,000 | 2,000 |
| Estimate of future 1981 expenses: | | | |
| TUGboat Vol. 2, Nos. 2&3: 800 copies | | | |
| Printing | $3,200 | | |
| Postage | 900 | | |
| Clerical labor | 200 | $ 4,300 | |
| Reserve for 1981 expenses for Cincinnati meeting, January 1982 | | 1,000 | |
| Support for Stanford TeX Coordinator[4] | | 3,600 | ( 8,900) |
| Subtotal: | | | $( 567) |
| Anticipated receipts in 1981 against 1982 individual membership (50% of membership) | | | 4,500 |
| Balance (estimate to December 31, 1981) | | | $ 3,933 |

*Notes:*

1. Total membership is 495, of which 30 are complimentary; of these, 371 members are domestic and 124 foreign.
2. The Implementors' Workshop held at Stanford, May 14–15, 1981, was attended by 92 participants.
3. Not included in these figures are costs for services provided by AMS professional staff, including programming, reviewing and editing, answering telephone inquiries, maintaining the mailing list, and other clerical services.
4. Professor Arthur Samuel is acting for Luis Trabb-Pardo as TeX coordinator, answering questions, distributing tapes, and fixing bugs in the TeX source code. Luis has asked, and the finance committee has agreed, that TUG contribute to Professor Samuel's support.

Respectfully submitted,

Samuel B. Whidden, Treasurer

* * * * * * * * * * *

## PROPOSAL FOR INSTITUTIONAL SUPPORT OF TUG

Robert Morris

Late last week (June 14) the Finance Committee met with an unusual opportunity to fund something which has made me change my previous position about TUG institutional membership. Barring an obstruction due to an Air Traffic Controllers strike, we are sending the chairman of TUG, Mike Spivak, to the ANSI standards committee on Text Processing Languages, X3J6.

This committee will be considering a number of possibilities for the processing of mathematical text, and one of the Steering Committee members, Lynne

Price, will be attending as a member. However, Lynne can not attend the beginning of the meeting and felt it important that TEX be represented at least informally by someone knowledgeable.

Acting in hastily convened and loosely organized telephone meetings, we agreed to pay the cost of Mike's attendance at this meeting as our observer, even though TUG has no funds in its budget beyond those needed to pay for the newsletter. Approximately $1000 will be borrowed from the AMS to be reimbursed from future TUG income.

In the Steering Committee meeting (see my minutes, page 4) it was agreed that we would propose no institutional dues until we had some proposal for use of such money to the benefit of the membership. Here is such a benefit: representation of the TEX user community at standards committees and other organizations which may be in a position to influence the use or restriction of text processing systems (for example, I could envision also presentations to governmental agencies who might be promulgating standards for government documents).

Another benefit I think should accrue to paying institutional members is an annual (?) tape of contributed macros and (perhaps) a copy of AMS-TEX when it is in its "positive versions" (in the current pre-release versions I am enthusiastic about distributing it at cost to anyone who wants to test it. Later, I would make it a benefit of institutional membership).

Thus I now argue for the following dues structure:

| | |
|---|---|
| Individuals | approximately the cost of TUGboat |
| Educational institutions | $100 |
| Non profit institutions using TEX in house | $250 |
| All others | $500 |

Note that I have included all commercial organizations and all users of TEX who use it to produce publications for sale (e.g. the AMS and university presses) as one class of users.

I hope the precise figures and the ratios will be the subject of much discussion in this forum, because I will ask for formal ratification of some such structure at the annual meeting in January.

I will collate any replies this note brings. Please mail them to me at:

- (before Sept 1): IBM Cambridge Scientific Center, 545 Technology Square, Cambridge, MA 02139.
- (after Sept 1): Dept. of Mathematical Sciences, UMASS/Boston, Boston, MA 02125.
- Arpanet address: ram@mit-mc.

If you are especially anxious that the full text of your reply be published in TUGboat, please so indicate.

*Editor's note: The X3J6 meeting described above has been rescheduled, and Lynne Price will probably attend rather than Mike Spivak, so that TUG funds will very likely not be required. Bob's new position, in favor of institutional support, is not affected by this change, a fact he has confirmed in a telephone conversation.*

* * * * * * * * * * *

## REPORT ON THE TEX IMPLEMENTORS' WORKSHOP, STANFORD, 14–15 MAY 1981

Barry C. W. Doherty

At the TEX Implementors' Workshop in May, 92 people were registered (a complete list follows). The goal was to draw together both those knowledgeable about TEX and those in various stages of the implementating TEX-in-Pascal, from having an interest to having completed the installation, so that there could be communication of the problems and solutions involved.

The first day consisted of a series of planned talks on various aspects of TEX, from advanced usage to desirable features of Pascal compilers and technical details of TEX's output. On the second day, a series of informal sessions focused on people's principal interests and concerns, attempting to provide the information most necessary for those trying to install TEX and to gather the major unsolved problems hindering such installation.

Some of the more 'formal' talks either appear as articles in this issue of TUGboat or will appear in subsequent issues. Similarly, a number of the topics addressed during the second day have generated communications that appear here. The range of interests was large, with the result that many participants felt that much more could (or should) have been said about each of the topics. (Perhaps these communications will stir such a discussion in these pages!)

### The schedule

**First day (May 14*th*)**

9:00–10:00 **Donald Knuth** *"TEX debugging aids"* A detailed analysis of sample TEX input using information available through features built into TEX (such as `\trace` and `\ddt`). It is hoped that a presentation of this talk will be available for the next issue of TUGboat.

10:00–10:45 **Ignacio Zabala** *"Pascal-related issues"* Concentration on the characteristics and suitability of various popular Pascal compilers, with suggestions on what to look for in a compiler and how to cope with the compiler one has. (*See this issue, p. 16.*)

11:15–12:00 **Ignacio Zabala** *"The system dependent module of TEX-in-Pascal"* The Pascal elements of TEX and their implications. (*See the articles by Lawson, Zabala and Díaz,* TUGboat Vol. 2, No. 2, pp. 20, 32.)

1:15–2:00 **David Fuchs** *"Different output formats, conversion issues"* Largely a discussion of TEX's DVI file format. (*See this issue, p. 12.*)

2:00–2:45 **Luis Trabb-Pardo** *"From DVI to paper"* General discussion of translator (driver) programs (from DVI to something a specific device understands), and the role of spoolers/servers in scheduling and queueing—features and characteristics, downloading of fonts, memory requirements, efficiency.

3:00–3:30 **Frank Liang** *"Hyphenation in TEX"* Discussion of the algorithm used in TEX and comparison with other widely used algorithms. (*See this issue, p. 19.*)

3:30–4:00 **Michael Plass** *"Lines, paragraphs, pages"* Discussion of how TEX functions in this context. See report by Donald E. Knuth and Michael F. Plass, *Breaking paragraphs into lines*, Stanford CSD report CSD-CS-80-828.

A panel discussion had been scheduled to begin at 4:00; talks were running longer than planned as a result of discussions following most. Instead Don Knuth spent a few minutes discussing his plans for TEX, which include a series of three books providing complete documentation on the system (dates are projected completion dates):

- TEX—an entire listing of the Pascal source code, a 'final' user manual, and a history of debugging TEX. (Winter 1982)
- Computer Modern Roman—a description of this font family. (Spring 1982)
- **METAFONT**—similar to the book on TEX. (Winter 1983)

. . . . . . . . .

**Second day (May 15*th*)**

9:00–10:00 *"TEX distribution and installation"* General problems of obtaining TEX and of the transportability of both TEX and TEX-related files. Questions were raised about the real utility of the current means of distributing TEX-in-Pascal as two quasi-independent documents (Pascal source code and internal documentation), both produced from the same meta-language source; general opinion seemed to favor distribution of the original source together with the programs (currently implemented only in SAIL) for producing the pieces, to allow each site to tailor the results to its (and its compiler's) needs more easily. One result was the formation of a tape standards committee. (*See the article by Milligan on this committee, p. 10.*)

10:00–11:00 *"***METAFONT** *and fonts"* Interest in both **METAFONT** and in the distribution of fonts. Again, one result was the formation of a committee to look into the problems. (*See the article by Doherty, p. 34.*)

11:00–12:00 *"Son of TEX"* Even before TEX's final release there have been numerous suggestions for what TEX might (or ought to) do. The spirit of these modifications is to allow more specialized typesetting to be done without damaging the compatibility with standard TEX. Some desired features include a more "suitable" input language, more tractable error messages, incorporation of graphics output, non-English hyphenation capabilities, batch mode (rather than interactive processing), and real-time interactive TEX. (*See the article by Price, p. 58.*)

1:00–2:00 *"Macro packages"* Already several major macro packages have been developed (see the documentation on the macro packages by Keller and Díaz, for instance, as well as Spivak's AMS-TEX, in various issues of TUGboat). Here there was an attempt to focus on standards and conventions of possible interest to macro writers: questions of compatibility, consistency in font-naming, conventions for replacing characters found on the Stanford non-standard terminal keyboards. (*See the articles by Milligan (p. 44) and Price (p. 43) in this issue.*)

2:00–3:00 *"Output devices and their interfaces"* A somewhat more specific examination of some of the more common output devices, their characteristics and what is required of their interfaces.

3:00–4:00 *"Architecture sessions"* About a half-dozen groups formed to discuss their particular problems. Major sessions included IBM, VAX, DEC 10s and 20s, CDCs.

4:00–5:30 *"Output device demonstrations"* This was devoted to Trabb-Pardo's presentation of the Canon Laser Printer (*see his article in this issue, p. 26*) and a tour of BNR given by Milligan (equipment including a Versatec, PERQ, and Alphatype).

* * * * * * * * * * * *

## Attendees, TEX Implementors' Workshop Stanford, May 14–15, 1981

Adamo, Vincent – Texas A & M University
Amabile, Carolyn – National Information Systems
Ash, William – Stanford Linear Accelerator Center
Ball, George – Washington State University
Beebe, Nelson – University of Utah
Beeton, Barbara – American Mathematical Society
Bennison, John – Brown University
Berns, Eagle – Stanford University
Blair, John – CALMA
Broadwell, Peter – Univ. of California, Santa Cruz
Brown, Malcolm – Stanford University
Buckle, Normand – University of Montreal
Bupara, Sarge – Exxon Office Systems
Carnes, Lance – Gentry, Incorporated
Chaffee, Roger – Stanford Linear Accelerator Center
Conley, Marsha – University of Illinois
Copeland, John
Cralle, Robert – Lawrence Livermore Lab
Dailey, William H. – Letterman Army Institute
Day, Christopher – Lawrence Berkeley Lab
Díaz, Max – Stanford University
Doherty, Barry – American Mathematical Society
Doob, Michael – University of Manitoba
Durling, Bob – University of California, Santa Cruz
Faul, Don – Lawrence Livermore Lab
Faulkner, Thomas – Washington State University
Forster, Doug – Stanford University
Frisch, Michael – University of Minnesota
Fuchs, David – Stanford University
Gittelsohn, Michael – San Francisco State University
Goldby, Alan – University of California, Santa Cruz
Grosso, Paul – University of Michigan
Guenther, Dean – Washington State University
Hickey, Thomas – OCLC, Incorporated
Hodge, Thea – University of Minnesota
Jackson, Calvin – California Institute of Technology
Katagiri, Grace – University of California, Berkeley
Kelley, Al – University of California, Santa Cruz
Knuth, Donald – Stanford University
Lanford, Oscar – University of California, Berkeley
Lindsey, Clark – University of California, Riverside
Mapes, Jeff – Stanford University
Melen, Randy – Stanford University
Milligan, Patrick – BNR, Incorporated
Morris, Bob – University of Massachusetts, Boston
Nichols, Monte – Sandia Labs
Norstad, John – Northwestern University
Nussbaum, Frank – Newline Graphics
Palais, Richard – Brandeis University
Payne, Thomas – University of California, Riverside
Pierce, Thomas – EG&G, WASC, Incorporated
Plass, Michael – Stanford University
Plass, Susan – Stanford University
Platt, Craig – University of Manitoba
Price, Lynne – BNR
von Raesfeld, Mary – National Information Systems
Reier, Warren – Gentry, Incorporated
Renz, Peter – W. H. Freeman and Company
des Rivières, Jim – Carleton University
Robb, Richard – Cemrel, Incorporated
Rosenschein, Jeffrey S. – Stanford University
Ross, Kenneth – University of Oregon
Rushworth, Tom – Block Brothers Industries
Sachs, Jonathan – independent contractor
Samuel, Arthur – Stanford University
Schechtman, Marty – Newline Graphics
Scott, Eric P. – California Institute of Technology
Sears, Chris – San Francisco State University
Sherrod, Phil – Vanderbilt University
Smith, Barry – Oregon Software
Spivak, Mike
Stovall, John – Wycliffe Bible Translations
Stromquist, Ralph – Univ. of Wisconsin-Madison
Tal, Avi – Electis Engineering Incorporated
Thedford, Rilla – Mathematical Reviews
Trabb-Pardo, Luis – Stanford University
Truax, Terry – Mathematical Reviews
Tuttle, Joey – I. P. Sharp Associates
Van Dalen, Gordon – University of California, Riverside
Van den Bosch, Peter – Univ. of British Columbia
Wakabayashi, Nobuo – Stanford University
Weening, Joe – Stanford University
Welland, Robert – Northwestern University
Wheeler, Norman
Whidden, Samuel – American Mathematical Society
Whipple, Edgar – Lawrence Berkeley Lab
Whitney, Lynn – Univ. of California, Santa Cruz
Whitney, Ron – American Mathematical Society
Wilmott, Sam – Block Brothers Industries
Wiser, David – Stanford Linear Accelerator Center
Wolf, Joe – University of California, Berkeley
Zabala, Ignacio – Stanford University

* * * * * * * * * * *

## PRELIMINARY ANNOUNCEMENT: TUG MEETING, CINCINNATI, JANUARY 1982

The next TUG meeting will be held in Cincinnati, Ohio, at the Stouffer's Cincinnati Towers from January 11–12, 1982. This meeting will review the growth and applications of TEX. All TUG members are urged to attend. There will be computer site dependent symposia as well as a general overview of TEX-in-Pascal. We hope also to have a demonstration of TEX.

A preliminary schedule will be mailed to TUG members early in the fall, as soon as a program has been devised. We would like to solicit reports on TEX implementation and usage. Discussion topics which are submitted by September 15 will be considered for inclusion in the preliminary schedule.

Please send such requests to:

Tom Pierce
TEX Users' Meeting
P.O. Box 880
Collins Ferry Road
Morgantown, WV 26505

* * * * * * * * * * *

## ASK NOT WHAT TUG CAN DO FOR YOU, ASK WHAT YOU CAN DO FOR TUG!

Patrick Milligan
BNR Inc.

At the recent TEX Implementors' Workshop, there were several discussions (both formal and informal) concerning the future of TEX and the TEX Users Group. The following article reflects my opinions about where we should be headed, and how we can get there.

It seems clear that the widespread acceptance and use of TEX is tied very closely to the success and growth of TUG. Without an effective forum for the interchange of ideas and information, TEX will probably not fulfill its potential as a standard language for computer typography. The TEX Users Group, through TUGboat, has begun to provide such a forum, but in order to function effectively, your assistance is required!

At the time of the Workshop in May, there were over 300 members of TUG. It is not known how many of this number are actual TEX users (as opposed to *potential* users awaiting a working implementation of TEX on their local computer facilities). In addition, it is not known how many TEX users have not yet become paying members of TUG. By definition, the TEX Users Group must have users of TEX in order to be a viable organization. Therefore, the primary goal of TUG should be to encourage and assist the growth of the TEX *user* community. There are several ways that you, as a member of TUG, can help:

1. If you are lucky enough to have a working TEX installation, encourage your local users to join TUG. In addition, share your experiences with the use and/or installation of TEX by sending letters, articles, bugs, and macros to TUGboat.
2. If you have received a version of TEX and are in the process of installing it on your local computer, let TUGboat know about your progress (or lack of progress). News of (temporary) failure is just as important as news of success!
3. If you are waiting for a version of TEX to be available on your flavor of computer architecture, contact your site coordinator to indicate your interest. In this way, you might be able to receive advance notice of a working TEX. Also, you might begin to acquire the necessary hardware for your output devices and begin to build some of the support software necessary to drive such devices.
4. If no one is implementing TEX on your flavor of computer architecture, obtain a copy of TEX-in-Pascal and begin your own installation effort. If you are not a systems programmer, you should be able to interest someone on your local computer staff to assist.

The intent of such communications to TUGboat is to minimize the "reinventing of the wheel". Each potential TEX installer should be able to draw upon a wealth of knowledge on the trials and tribulations of TEX installation. Each novice macro writer should have numerous examples available to learn from. It is frustrating to hear second-hand rumors at TUG meetings or workshops like: "So and So at SRI has a working VAX/UNIX TEX" or "Someone at DEC has a Diablo device interface" or "Somebody at MIT has some nice thesis macros." Just as Don Knuth has shared TEX with the world, it is imperative that you share your TEX experiences with TUG.

Many of TUG's current problems are due to a lack of "critical mass". The porting of Pascal TEX to many architectures, and the availability of output devices and their interfaces has not happened as quickly as anticipated. At the TUG Steering Committee meeting in May, the issues of institutional memberships and TEX support were discussed, but not resolved. The primary obstacles to the institutional memberships were (a) the fear that such fees would inhibit the installation of TEX by small organizations or universities, and (b) the current organization of TUG does not easily allow additional services beyond TUGboat as an enticement to make such fees worthwhile. The bottom line seems to be that there aren't enough TEX installations willing or able to bear the burden of additional services such as TEX support or enhancement. As the number of TEX-in-Pascal installations grows, the direction and functions of TUG will grow also.

Once the first hurdle of providing TEX to a wide base of users is met, there are other challenges for TUG to face. In the area of output device support, there is a strong need for portable device drivers and TEX support tools written in Pascal or some other widely used programming language. Admittedly, standard Pascal does not provide the full set of facilities required to write such device drivers, but most Pascals provide some means of escape or extension to allow full use of the underlying operating system. It would be a useful exercise in portability if large portions of device driver code were written in standard Pascal, with architecture or operating system dependences collected together in one or more system dependent modules (like the SYSDEP code of TEX-in-Pascal). One example of such a program is the Pascal version of DVITYP, written by David Fuchs at Stanford. Pascal TEX itself is an interesting experiment in portability. These examples are just the beginning; much more work needs to be done in this area.

Another direction for TUG growth is in the area of macro packages. Most TEX installations quickly discover that one or more layers of macros are required to insulate their users from "naked" TEX. Many useful macro packages have been presented in TUGboat. Michael Spivak's comprehensive AMS-TEX macros have been thoroughly documented in *The Joy of TEX*. However, many more useful and interesting macros have been developed but not contributed to TUG. Also, the issue of portability is applicable to macro packages as well: the use of extended ASCII character sets, font codes, counters and boxes all make the job of merging several macro packages together difficult. Output device dependences may find their way into macros, thus defeating TEX's "device independent" output. It is hoped that Lynne Price, the TUG macro coordinator, may be able to bring some order out of chaos in this area (with your help). Awareness of the portability and modularity issues will assist TEX macro writers; standards and conventions encouraged by TUG will also help.

Closely related to the issues of portable TEX support tools and macro packages is the area of machine readable distribution. A proposed standard for machine independent tape interchange is discussed elsewhere in this issue of TUGboat (page 10). Stanford has attempted to solve this problem for the distribution of TEX-in-Pascal, macros, and fonts. The current organization of site coordinators has solved the problem of distribution between sites using similar computers, through the use of common, operating system dependent tape formats. However, the problem of general, machine independent tape interchange between TEX users who use different computers has not been completely solved. It is important that standards for tape interchange be established, and portable tools developed to support these standards.

One potential area which TUG should explore is the sale of machine readable macros and programs submitted to TUGboat. Having one distribution center for these contributions would be preferable to contacting the author(s) of a particular program or macro package. Receiving one tape from TUG would be easier than requesting tapes from multiple sources, and would be much easier than typing in part or all of a long macro package or program. In addition, TUG would have another source of revenue! This sort of scheme has worked well for the DECUS Library (a part of the DEC Users Society), and for Addison-Wesley's distribution of Ratfor source for the programs in Kernighan and Plauger's *Software Tools*.

In conclusion, it is clear that what you get out of TUG depends on what you are willing to put into it! Without member contributions, there would be no TUGboat. Without volunteers, there would be no TUG Steering Committee. The future for TEX and TUG looks bright, provided we can ease our growing pains (with your help). Before I step down from my soap box, I would like to thank all of you who have made the TEX Users Group and TUGboat possible through your involvement. The staff of the American Mathematical Society deserve special thanks for their hard work and patience.

* * * * * * * * * * *

## A PROPOSAL FOR A MACHINE INDEPENDENT TAPE INTERCHANGE STANDARD

Patrick Milligan
BNR Inc.

At the TEX Implementors' Workshop in May, a committee was formed to propose a tape format suitable for machine independent and operating system independent interchange of TEX source files. The members of this committee are:

| | |
|---|---|
| Nelson Beebe | University of Utah |
| Patrick Milligan | BNR Inc. |
| Robert Morris | UMASS/Boston |
| Susan Plass | Stanford CIT |

The motivation behind this proposal is to provide a means of submitting machine readable TEX source to TUGboat (and someday to AMS journals), as well as a means of distributing and exchanging TEX macros and manuscripts. To some extent, the problem of tape interchange formats has been addressed by TUGboat in its ASCII "card image" format (80 characters/record × 100 records/block). The primary problems with such a format stem from TEX's use of the full ASCII character set. The following potential problems exist:

- Not all computer systems support the ASCII character set, and those that do may limit or prohibit the use of ASCII control characters. There are "standard" translations between ASCII and EBCDIC graphic characters, but no such translations exist for control characters. TEX can usually avoid the use of control characters, but as we have seen in recent TUGboat macro packages and in the TEX manual itself, it is tempting to use the "extended" ASCII character sets in use at Stanford, MIT, and CMU if they are available. In addition, TEX's control sequences for negative conditional thin space (\≤) and conditional thin space (\≥) *must* be entered using control characters!

- TEX makes some assumptions about the underlying structure of text files. In particular, it is assumed that a file is organized as a long string of characters which is divided into lines by end-of-line characters, and into pages by form-feeds. On some systems, the structure of text files is either fixed length "card image" records, padded with blanks (and possibly with sequence numbers in columns 73–80), or variable length records rounded to computer word boundaries and padded with blanks or some other filler. In most cases, it is not important to know where the placement of the end-of-line is, or whether the trailing blanks on a line are "real" or supplied by the system. However, if the meaning of blanks or end-of-line characters is changed through the use of the \chcode control sequence, their placement and existence becomes critical. Many powerful techniques presented at the TEXarcana mini-course depend on the ability to redefine space or carriage-return to invoke a control sequence. Arthur Keller's \nofill macro (presented in TUGboat Vol. 2, No. 1) also uses this feature of TEX.
- Another attribute of some text file representations is limited line length. The worst case seems to be the fixed width card image format with sequence numbers. Since TEX allows lines up to 150 characters, unless care is taken, TEX source may overflow the 72 character limit imposed by some systems. Even if a conscious effort is made to limit line length, there are times when it is difficult if not impossible to break a line for fear of introducing a significant space. For example, the \qspace macro in AMS-TEX has one line which is 98 characters long, and it can't easily be broken since the space character has been redefined to be category 12 via \chcode.

Many of the problems listed above must be resolved in the system dependent module of Pascal TEX for each architecture. By definition, our tape interchange format must be independent of the design decisions that were made for a specific implementation of TEX. The best we can do is provide a format that can be transformed into suitable input for Pascal TEX on a given system. It is also hoped that such a transformation is reversible. An additional constraint placed on our tape format is that it should be able to accommodate TEX source containing control characters, significant trailing spaces and carriage returns, and long lines. It is not our place to pass judgment on the use of TEX's somewhat esoteric tricks: We must accept the reality that such features will be used.

In order to meet our constraints of machine independence and compatibility with TEX's idealized notions of text files, we are proposing a tape format which represents a TEX source file as a stream of ASCII characters separated into lines by carriage-return linefeed pairs. This stream of characters will be broken into tape records $N$ bytes long, where $N$ will be chosen such that (1) a tape record will exactly fill an integral number of words on all targeted architectures and (2) $N$ will be large enough to effectively utilize the tape. Suggestions for a good value of $N$ would be greatly appreciated! The last block of the tape should be padded with NULs. In order to avoid problems with "helpful" systems that like to throw away "unwanted" characters, each ASCII character will be represented as two hexadecimal digits.

In order to make this format work, each TEX installer for a given architecture will have to write two programs: One to read such a tape and transform the data into a machine-dependent text file format that TEX will digest, and another program to perform the reverse transformation and output a hex-encoded tape. The design decisions that went into the implementation of the system-dependent module for Pascal TEX will be applicable to these tape utilities.

It is assumed that 9-track tapes will be used, although the hex encoding would work equally well for 7-track tapes (using a 6-bit ASCII subset for each digit). The same coding scheme can be used to transfer files over phone lines if $N$ is chosen to be a reasonable terminal line length.

An added benefit to this format is that it can be used to transfer binary data such as DVI, TFM, and font files with few modifications. In this case, the two hex digits would represent an 8-bit data byte instead of a 7-bit ASCII character.

It seems clear that we need a tape standard that addresses the problems of machine independent information exchange, while still providing the functionality that TEX requires. There are two questions to be asked:

1. Is this the format that we need?
2. Is it worth the effort involved?

Your input is needed to answer these questions. Feedback from those of you who have been actively working on porting Pascal TEX to new architectures is especially welcome. Please respond!

* * * * * * * * * * *

## Software

* * * * * * * * * * *

### THE FORMAT OF TEX'S DVI FILES VERSION 1

David Fuchs
*TEX Project, Stanford University*
*April 18, 1981*

When TEX compiles a document, it produces an output file that contains specifications of how TEX has decided the formatted text should appear in hard copy. These output files are known as '.DVI' files, which stands for 'device independent'. For instance, running TEX and telling it to \input dviinf will cause TEX to look for a file called DVIINF.TEX, read it, and produce an output file called DVIINF.DVI, which is a .DVI file. This document describes the format of .DVI files in detail, giving all the specifications along with examples.

A .DVI file contains information about where characters go on pages. The format is such that there are those who say that almost any reasonable device can be driven by a program that takes .DVI files as input. In particular, a .DVI file can be printed on the Xerox Dover, Xerox Graphics Printer (XGP), Varian, Versatec, Canon and Alphatype at the Stanford CS Dept., depending on what spooler it is passed to.

The .DVI file is a stream of 8-bit bytes, packed in computer words high-order byte first. If the computer word length is not evenly divisible by 8, then the extra bits at the low-order end of each word will be unused. The first byte in a .DVI file is byte number zero, the next is number one, etc. For example, on Stanford's 36-bit word machines, byte number 0 is in the highest order eight bits of the first word in a .DVI file, while byte number 7 is in the twelfth through fifth least significant bits of the second word in the file; and the least significant four bits in every word are zero.

A .DVI file is actually a series of **commands**. A **command** consists of one byte containing the **command**'s unique number, followed by a number (possibly zero) of **parameters** to the **command**. A given **command** always has the same number of **parameters**. These **parameters** may take from one to four bytes each, but a given **parameter** of a given **command** always takes the same number of bytes. Some **parameters** may sometimes be negative, in which case two's complement representation is used. The complete list of **commands**, with a description of all the .DVI **commands** and their **parameters**, is below. The reader is encouraged to refer to the **command** list while reading the various examples in this document.

In the **command** descriptions, a lower case letter with a [bracketed] number following it means that the **command** has a **parameter** that is that number of bytes long. An X3 **command**, for instance, is 3 bytes long, the first byte of which has the decimal value 144, the second and third of which give the distance to move to the right. If the second byte $= S$ and the third $= T$, then the distance to move is $2^8S+T$ (but if the high order bit of $S$ is a one, then the distance to move is $2^8S+T-2^{16}$, considering $S$ and $T$ as being in the range [0..255]).

The .DVI file contains a number of **pages** followed by a **postamble**. A **page** consists of a BOP **command**, followed by lots of other **commands** that tell where the characters on the page go, followed by an EOP **command**. Each EOP **command** is immediately followed by another BOP **command**, or by the PST **command**, which means that there are no more **pages** in the file, and the remaining bytes in the .DVI file are the **postamble**. Remember that TEX really doesn't have an official knowledge of **page** numbers (although it does print the value of \count0 on your terminal as it outputs each **page** on the assumption that some meaningful number is there), so the only thing that can be said about the ordering of pages in a .DVI file is: The order in which **pages** come in a .DVI file is the same order in which TEX constructed them, which is the same order in which the TEX user specified them. Any blank or nonexistent **page** from a TEX job might not be in the .DVI file at all. If we consider the **page** number to be the value of \count0, then the **page** following **page** number 34 in a .DVI file might well be **page** number —5.

Some **parameters** of .DVI **commands** are **pointers**. A **pointer** is simply a byte number as discussed above. A **pointer** itself is 4 bytes long. For example, a BOP **command**'s last **parameter** (p[4]) is the BOP's **previous page pointer**. This parameter is the number of the byte in which the previous **page**'s BOP command begins. In particular, the *second* **page**'s BOP **command**'s **previous page pointer parameter** (p[4]) is always zero, since the first **page**'s BOP is always in byte zero in a .DVI file. If the first **page** in a .DVI file had only a BOP and EOP **command**, then the *third* **page**'s BOP's **previous page pointer** would be 46, since the first **page's** BOP **command** takes bytes zero through 44, the first **page's** EOP is byte 45, so the *second* **page's** BOP is in byte 46.

When a `.DVI`-reading program reads the **commands** for a **page**, it should keep track of the **current font**. This can be done with a single integer variable, the value of which will always lie in the range $[0..2^{32}-2]$. The value of the **current font** is changed only by FONT and FONTNUM **commands**. Whenever a **command** occurs in the `.DVI` file that causes a character to be set on the **page**, the character is implicitly from the **current font**.

Likewise, the program should keep track of the **current position on the page**. The **current position on the page** is like a cursor on the **page**; whenever a character or rule is set, it gets put at the **current position on the page**. The **current position on the page** is just two numbers—which are called **horizontal coordinate** and **vertical coordinate**. Moving to the right on a **page** is represented by an increase in **horizontal coordinate**, while moving down is an increase in **vertical coordinate**. The upper-left-hand corner of the **page** is **horizontal coordinate** = **vertical coordinate** = 0 (i.e., our system is slightly non-cartesian). Both **coordinates** are given in rsu's (ridiculously small units), where 1rsu $= 10^{-7}$meter. This is so that accumulated errors will be insignificant even in the worst imaginable case (a "box" many feet long). The **current position on the page** is moved about by the commands W0, W2, W3, W4, X0, X2, X3, X4, Y0, Y2, Y3, Y4, Z0, Z2, Z3 and Z4: The **vertical coordinate** is changed by Y and Z **commands**, while the **horizontal coordinate** is changed by W and X **commands**. (The value of **horizontal coordinate** can also change as a side effect of setting a character or rule (VERTCHAR and VERTRULE **commands**)—the **current position on the page** moves right the natural width of the character or rule set. The POP **command** may also change **current position on the page**.)

So, whoever or whatever reads a `.DVI` file might have three variables, $F$, $H$ and $V$, to keep track of the **current font** and the **current position on the page**. Four more variables are also called for: **w-amount**, **x-amount**, **y-amount**, and **z-amount**. These variables hold not locations, but distances (in rsu's). The **amount** variables are used in `.DVI` files to move the **current position on the page** around: The **commands** X0 and W0 add **x-amount** and **w-amount** to **horizontal coordinate**, respectively, while Y0 and Z0 add **y-amount** or **z-amount** to **vertical coordinate**, respectively. There are also a number of **commands** that change the value of **w-amount**, **x-amount**, **y-amount** or **z-amount** (W2, W3, W4, X2, X3, X4, Y2, Y3, Y4, Z2, Z3 and Z4; these commands also change **horizontal coordinate** or **vertical coordinate**). Actually, the `.DVI`-reading program must have a stack that can hold **horizontal coordinates** and **vertical coordinates**, as well as **w-**, **x-**, **y-**, and **z-amounts**. These six values always get pushed and popped together, and a reasonable maximum stack depth might be about 200 (times six, since six items get pushed at once). As each **page** starts, a `.DVI` reading program should set the **amount** variables to zero. The stack should be empty. The initial value of $F$ doesn't matter, since *every* **page** of a `.DVI` file must have a FONT or FONTNUM **command** before any **command** that will set a character (the HORZCHAR and VERTCHAR **commands**). Note that $F$ is *not* pushed and popped.

A program called `DVITYP` is available that takes any `.DVI` file and prints a readable description of its contents, together with error messages if the file is not in the correct format.

. . . . . . . . .

Command Name
 Command Bytes
 Description

VERTCHAR0
 0
 Set character number 0 from the **current font** such that its reference point is at the **current position on the page**, and then increment **horizontal coordinate** by the character's width.

VERTCHAR1
 1
 Set character number 1, etc.

⋮ ⋮

VERTCHAR127
 127
 Set character number 127, etc.

NOP 128
 No-op, do nothing, ignore. Note that NOPs come *between* **commands**, they may not come between a **command** and its **parameters**, or between two **parameters**.

| | | |
|---|---|---|
| BOP | 129 c0[4] c1[4] ... c9[4] p[4] | Beginning of page. The parameter p is a pointer to the BOP command of the *previous* page in the .DVI file (where the *first* BOP in a .DVI file has a p of −1, by convention). The ten c's hold the values of TeX's ten \counters at the time this page was output. |
| EOP | 130 | The end of all commands for the page has been reached. The number of PUSH commands on this page should equal the number of POPs. |
| PUSH | 132 | Push the current values of horizontal coordinate and vertical coordinate, and the current w-, x-, y-, and z-amounts onto the stack, but don't alter them (so an X0 after a PUSH will get to the same spot that it would have had it had been given just before the PUSH). |
| POP | 133 | Pop the z-, y-, x-, and w-amounts, and vertical coordinate and horizontal coordinate off the stack. At no point in a .DVI file will there have been more POPs than PUSHes. |
| HORZRULE | 135 h[4] w[4] | Typeset a rule of height h and width w, with its bottom left corner at the current position on the page. If either $h \leq 0$ or $w \leq 0$, no rule should be set. |
| VERTRULE | 134 h[4] w[4] | Same as HORZRULE, but also increment horizontal coordinate by w when done (even if $h \leq 0$ or $w \leq 0$). |
| HORZCHAR | 136 c[1] | Set character c just as if we'd gotten the VERTCHARc command, but don't change the current position on the page. Note that c must be in the range [0..127]. |
| FONT | 137 f[4] | Set current font to f. Note that this command is not currently used by TeX—it is only needed if f is greater than 63, because of the FONTNUM commands below. Large font numbers are intended for use with oriental alphabets and for (possibly large) illustrations that are to appear in a document; the maximum legal number is $2^{32} - 2$. |
| X2 | 144 m[2] | Move right m rsu's by adding m to horizontal coordinate, and put m into x-amount. Note that m is in 2's complement, so this could actually be a move to the left. |
| X3 | 143 m[3] | Same as X2 (but has a 3 byte long m parameter). |
| X4 | 142 m[4] | Same as X2 (but has a 4 byte long m parameter). |
| X0 | 145 | Move right x-amount (which can be negative, etc). |
| W2 | 140 m[2] | The same as the X2 command (i.e., alters horizontal coordinate), but alter w-amount rather than x-amount, so that doing a W0 command can have different results than doing an X0 command. |
| W3 | 139 m[3] | As above. |
| W4 | 138 m[4] | As above. |
| W0 | 141 | Move right w-amount. |
| Y2 | 148 n[2] | Same idea, but now it's "down" rather than "right", so vertical coordinate changes, as does y-amount. |
| Y3 | 147 n[3] | As above. |
| Y4 | 146 n[4] | As above. |
| Y0 | 149 | Guess. |
| Z2 | 152 m[2] | Another downer. Affects vertical coordinate and z-amount. |

| | |
|---|---|
| Z3 | 151 m[3] |
| Z4 | 150 m[4] |
| Z0 | 153<br>Guess again. |
| FONTNUM0 | 154<br>Set **current font** to 0. |
| FONTNUM1 | 155<br>Set **current font** to 1. |
| ⋮ | ⋮ |
| FONTNUM63 | 217<br>Set **current font** to 63. |
| PST | 131 p[4] n[4] d[4] m[4] h[4] w[4]<br>Fontdef Fontdef ... Fontdef<br>-1[4] q[4] i[1] 223[?]<br>The **postamble** starts here. See below for the full explanation of the **parameters** of the **postamble**. |

Commands 218–255 are currently undefined and will not be output by TEX.

The PST **command**, which is always the last command in a `.DVI` file, is somewhat special. The **parameter** p is a **pointer** to the BOP of the final **page** in the `.DVI` file. The **parameters** n and d are the numerator and denominator of a fraction by which all the dimensions in the `.DVI` file should be multiplied by to get rsu's (TEX always outputs a 1 for each of these values, they are included in `.DVI` format to allow other text systems to conveniently output `.DVI` files). The **parameter** m is the overall magnification requested by par12 in the TEX job (par12 is unitless, and is 1000 times the desired magnification). Next come h and w, which are the height of the tallest **page**, and the width of the widest (both in rsu's).

Next in the postamble come the **font definitions**, one for each font used in the job (i.e., each FONT and FONTNUM **command** in a `.DVI` file must refer to a **font number** that has a **font definition**). The format of a **font definition** can be considered to be:

`fnum[4] fchk[4] fmag[4] fnamlen[1] fnam[fnamlen]`

The **font number** is held in `fnum`. The **font checksum** (from the font's TFM file) is in `fchk`. The **parameter** `fmag` holds the **font magnification** (1000 times the 'at size' of the font divided by its 'design size' (or just 1000 if there was no 'at' specification for the font)). Next comes the byte `fnamlen`, which is the number of characters in the **font name**, followed by the the **font name**, one ascii character per byte (right justified). Note that the font name includes a directory only if the font is not in the standard default library directory. From the definitions of the **parameters** of the PST command, note that the end of the **font definitions** is marked by a **font number** of -1 (which is *not* a legal font number). The four bytes following this phony **font number** constitute the **parameter** q, which is a **pointer** to the PST **command** (i.e., the beginning of the postamble). Next is a single byte **parameter** i (called the ID byte). Currently, the ID byte should always have a value of 1; it will be changed to 2 on the next incompatible release of `.DVI` format in 1990. Finally, there is some number (at least 4) of bytes whose value is 223 (base ten = '337 octal).

The idea of the q **pointer** at the end of the **postamble** is that a `.DVI` reading program can start at the end of the `.DVI` file, skipping backwards over the 223's, until it finds the ID byte. Then it can back up 4 bytes, read q, and then do a random seek to that byte number within the `.DVI` file. Now the **postamble** can be read from start to finish, while storing away the names and magnifications of all the fonts. Now the program can jump to the start of the `.DVI` file and read it sequentially. The reason for reading the **postamble** first is that to figure where the characters on a page go, the `.DVI` reading program must know the widths of the characters (see the VERTCHAR commands' description above). To find the widths, the `.DVI` reader must know the names of the fonts so it can get their widths from a TFM or VNT (or some other kind of font) file. But TEX can't put out all the font names until the end of the `.DVI` file because new fonts can appear anywhere in the TEX job. If **font definitions** were scattered throughout the `.DVI` file, then a spooler that read `.DVI` files would have to read all the pages of the `.DVI` file, even if the user only wanted the last page printed. The decision to put the **font definitions** in the **postamble** was based on these considerations, and the fact that just about any reasonable systems language allows random access. Unfortunately, standard PASCAL does not offer this feature. If it is absolutely necessary for a `.DVI` reading program to be written in standard PASCAL, then it either must make two passes over the `.DVI` file, or TEX must be doctored to output *two* files: the regular `.DVI` file, plus a PST file, which contains only the **postamble**. So far, there have been no reports of any installation of TEX that required this kind of kludge.

A few words on magnification: If you have a TEX document that does not mention any 'true' dimensions, then if you change just its **\magnify** statement, the .DVI file produced by TEX will change in just *one* place—the word in the postamble that records the requested magnification. The idea is that any spooler that reads the .DVI file will multiply *all* dimensions in the .DVI file by the magnification, thus the default magnification in the .DVI file may be easily overridden at spooling time. So, if the document specifies **\magnify**{1200}, a **\vskip 34cm** will be recorded in the .DVI file as $.34 \times 10^7$ rsu's of white space, but the spooler will multipy this by 1.2, making 40.8 centimeters of white space on output. If the user tells the spooler to use a magnification of 1000 rather than the 1200 in the .DVI file, then the output will have 34cm of white space. If a dimension in the document is specified as being 'true', then TEX divides the distance specified by the prevailing magnification, so that when a spooler looks at the .DVI file and multiplies by the magnification, it gets back the original distance. So, if we \vskip 24truecm while the magnification is 1200, TEX puts out .DVI **commands** that specifies 20 centimeters of white space. An output spooler that reads this .DVI file then puts $20 \times 1.2 = 24$cm of white space on its output. Of course, 'true' dimensions will come out 'false' if the spooler is told to override the magnification.

Font magnification goes one step further. Assume for a moment that the overall magnification is 1000. Now, if a TEX job specifies \font A=CMR10 at 15pt, say, that font's magnification is recorded as 1500 in its **font definition**. When a spooler reads this .DVI file, it will try to use the file CMR10.150VNT (or CMR10.150ANT, depending on the device), which is just like CMR10.100VNT, but the dimensions of all its characters were multiplied by 1.5 before they were digitized. An uppercase 'W' in CMR10 is 10pt wide, but CMR10 at 15pt has a 15pt wide 'W', so after VERTCHAR87 is seen, **horizontal coordinate** is increased by (15pt) $\times$ (254000rsu/72.27pt). Overall magnification is taken into account after all other calculations; for example, at magnification 1200 the font CMR10.120VNT would be used. Note that if the user had asked for cmr10 at 15truept, the factors would cancel out so that CMR10.150VNT would be the font chosen regardless of magnification. The magnification factor is given times 100 in the font file name so that roundoff error due to several multiplications will not affect the search for a font with characters of the right size. This convention about font file names is merely a suggestion, of course, it is not part of the .DVI format per se.

. . . . . . . . .

**Appendix: Comparison between version 0 and version 1.**

Note that .DVI files have an ID byte at the end of the **postamble**, which tells what version they are. The changes since version 0 are:

DVI files now use the *upper* bits in a word on machines whose word size isn't evenly divisible by 8. The BOP command has *ten* \counter parameters. The size of rsu's has changed to be $10^{-7}$meter. The postamble has changed to include overall magnification as well as a fraction that allows use of non-rsu dimensions. Font checksum and magnification are new, as is the convention about default directory name. Font descriptions in the postamble give the length of font names rather than delimiting them with a quoting character. The old zero ID byte is now a one.

**Some ideas for version 2.**

Although 1990 is still a ways off, we are currently expecting that version 2 of .DVI files will differ in the following ways:

The ID byte will be 2. The q bytes of the postamble will be preceded by 's[2]' where *s* is the maximum stack depth (excess of pushes over pops) needed to process this file.

* * * * * * * * * * *

## SOME FEEDBACK FROM PTEX INSTALLATIONS

Ignacio Zabala

The Pascal version of TEX was designed and written with the intent to generate a transportable program. Nevertheless, given the characteristics of the TEX system, some special assumptions had to be made about the Pascal environment in which PTEX was to be installed. Essentially, the requirements are:

- The system should have enough addressable memory to store the large arrays employed by PTEX (about 128K words of 32 bits).
- The compiler should be able to really pack fields of a PACKED RECORD and overlap multiple variants of packed records. If this requisite is not satisfied, PTEX will require at least four times as much memory.
- The compiler should be able to handle large case statements (say over 64 actual cases in a range [-500..500]) and have a default case (this is non-standard in Pascal but available in most compilers).

Additionally, PTEX requires an EXTERNAL (or separate) compilation facility. If no such thing is available, the SYSDEP module has to be inserted both in TEX and in TEXPRE by hand. Also, if there is no compile time variable initialization, the INITPROCEDURE appearing in the program has to be changed into an ordinary procedure.

Even though we tried to avoid it, the fact that PTEX was developed and debugged on a PDP-10 with Hamburg Pascal influenced the way the program was coded and documented. This compiler was often permissive in the same way as other languages of common use in Stanford (SAIL). Only feedback from other installations can help us improve the transportability of the program.

We have been lucky in receiving information from people who really worked on (and reported) both errors in the program and incompatibilities in the compilers.

The following are some of the problems that other compilers have had with the system. As said, it is often the case that the difficulty is due to the permissiveness of our Pascal, and not to the installation's compiler:

- Source must be all uppercase. (CD Cyber)
- Tab characters not allowed in the source. (P8000)
- Identifiers should be different in the first 8 characters. (VS, UW, P8000)
- Identifiers longer than 15 characters will not be accepted. (VAX)
- No octal ("20b") notation. (VS, P8000)
- All declared labels must be used. (VAX, VS, UW)
- Can't take large procedures. (VS, UW, P8000)
- Can't take large arrays. (MULTICS)
- No standard MAX and MIN functions. (P8000)
- Cannot take fields of packed records as actual parameters. (VAX, VS)
- Argument to PACK must be of type `array` (it's not enough that it evaluates to array). (VS)
- Loop counters must be local variables. (VS)
- Labels and gotos must be local to same block: cannot go to a label inside the `else` part of an `if` statement from inside the body of the true branch. (VS)
- No nested WITH statements allowed. (UW)
- Requires ENVIRONMENT modules for external linkage. (UW)
- Variables must be initialized before their use. They are not cleared by default. (VS)
- No GOTO labels in enclosing procedures. (UW)
- No INITPROCEDURE. (VAX, UW, MULTICS)
- Can't take large CASE statements. (UW)
- No EXTERN procedures. (P8000)
- Instead of OTHERS: the default case of CASE statements is:
  ELSE: (P8000)
  OTHERWISE: (VAX, CD Cyber)
  OTHERWISE (UW)
  None (MULTICS, SUNY)
- Can't pack `memoryword` properly. (CD Cyber)
- In packed records, elements defined of type 0..255 or 0..65535 are stored in whole 32 bit words. Records are assigned to length of the longest freevariant possible. (P8000)

All reports have received due attention. Currently, the code is all uppercase in lines that are never longer than 72 characters. All identifiers are shorter than 16 characters and differ in the first 8 characters. Octal variables appear only in the module that contains the system dependencies.

Two more particularly interesting problems are worth mentioning here.

Eagle Berns, while running PTEX with PASCAL-VS, detected a case statement for which no default had been provided, and whose switch variable was out of range. Intendedly, execution should have resumed after the case statement and that is what Hamburg Pascal did. PASCAL-VS signalled an error. Unfortunately, this situation is left undefined in the Pascal report.

Bill Kelly, using UW Pascal, detected trouble in the statement `pagemem[curchar]:=scanlength;` The function `scanlength` has the side-effect of changing `curchar`. UW Pascal (as opposed to Hamburg Pascal) does not evaluate subscripts on the left side of the assignment until the right side has been evaluated.

The original SAIL program assumed that variables would be implicitly initialized to 0, and the assumption was still valid for our Pascal. Much work had to be put into initializing everything before its use.

Below, we present a synthesis of some of the reports that have been most helpful in our project.

MOORE SCHOOL: UNIVAC SERIES 90 — PASCAL 8000 (GEORGE OTTO)

Pascal 8000/1.2 does not accept numbers like 100B or 400000B. These numbers must be changed to the appropriate integer or real form.

Pascal 8000/1.2 does not support EXTERN procedures and functions because of the internal loader.

Pascal 8000/1.2 uses ELSE: for the default case of CASE statements. (Not OTHERS: like Hamburg Pascal).

Tab characters not allowed in source.

Our Pascal *must* uniquely distinguish between all identifiers in the first 8 characters. Longer identifiers can be used, but only the first 8 characters of them are significant!

At the moment we are having trouble writing a tape from our EBCDIC machine to be read by Wharton's ASCII machine, to be sent to you over the net.

No standard `MAX` and `MIN` functions.

Pascal 8000/1.2 has a problem recognizing 10000000000.0 as a real. The fix is to use 1.0E10, instead.

Pascal 8000/1.2 stores elements defined 0..255 and 0..65535 in 32 bit words. Records are assigned to length of the longest freevariant possible. Therefore, the memory structures of TEX will not work as is.

U. OF MINNESOTA: CD CYBER (MIKE FRISCH)

- Everything must be uppercase
- Can't pack `memoryword` properly (this is bad)
- Had to replace `OTHERS:` by `OTHERWISE:`

JET PROPULSION LAB: UNIVAC 1100/81 — U WISCONSIN PASCAL (CHARLES LAWSON)

- This compiler employs `environment` modules (CD made one containing outer block `TYPE` and `EXTERNAL` procedure declarations)
- Found inconsistent definition and use of `ReadFontInfo` arguments.
- Changed `INITPROCEDURE` to ordinary procedure, and deleted empty block at end of SYSDEP.
- Changed `OTHERS:` to `OTHERWISE`.
- Changed type of `brchar` from `INTEGER` to `AsciiCode`.
- This compiler does not allow `GOTO` labels in enclosing procedures: in `quit` changed `GOTO 100` by a comment.
- Deleted unused labels.
- Found nested `WITH` curinput in getnext.(twice)

MULTICS: BENSON MARGULIES

The compiler dislikes the construction `INITPROCEDURE`. There is an array that is claimed to be too big. (May be solvable.) Impossible to deal with the need for an `OTHERWISE` statement, which the compiler does not provide. The filename interface of PTEX is still basically PDP10 oriented. For a machine without a fixed number of "channels" the file opening interface is problematic, requiring the establishment of an arbitrary limit.

U OF WISCONSIN: UNIVAC 1100/82 — U OF WISCONSIN PASCAL (BILL KELLY)

A major problem in converting TEX for the 1100 has been the differing methods of external compilation. In UW Pascal, all global declarations, including procedure and function heads must be included in an "environment module".

It would be helpful if the same names were used for the same types in both TEX and SYSDEP. When we received TEX, a type might be called `packedhyphenbit` in one and `pckdhyphbits` in the other. Our compiler does not accept identically defined but differently named types as identical in procedure parameters.

I was a bit confused by the `INITPROCEDURE` business at first. the documentation ought to say a bit more about this: namely, that that syntax allows compile-time initialization on your compiler, that it should be changed into a procedure in compilers without this feature, and where it should be called in TEX and TEXPRE.

We have a problem in the compiler with large case statements. It does not handle statements with a large number of cases, and the case statement in `maincontrol` gave some problems with this. There isn't a fixed limit in the compiler, but I broke the case statement in two, and the compiler had no problem.

The sheer size of TEX has given us some problems. The UNIVAC's instruction set includes many instructions with a 16-bit address field that can only address 64K of data. The data area for TEX runs to something like 71K for us, and we had to cut `mem` down from 32K to 25K to get the compiler to accept it. This would have been easier if a Pascal version of UNDOC were available, or if UNDOC had left `memsize` as a named Pascal constant instead of reducing it to 32767, and `memsize-1` to 32766, etc. I had to go through with a text editor and locate all references to 32767 and 32766 and determine by comparing the Pascal listings against the printed TEX listings whether these were actually references to `memsize`. I seem to have gotten them all because we haven't had subscript out of range errors, but it did mean that all the memory reduction was from the higher end of `mem` which is probably not optimal. We occasionally run into "`TEX capacity exceeded: memsize=25000`". I didn't try to alter the other memory parameters like `varsize` because there were so many instances of `varsize+1` and such that would have been affected.

We ran into another interesting problem: on a UNIVAC, a person typing at a terminal can type "`@eof`" and his terminal input is considered to have reached an end of file. This concept doesn't exist on most systems, so it wasn't considered in TEX. Basically, if a person types "`@eof`", I artificially return "\end" to TEX, but this doesn't always work.

I need to do more work on this. If this affects other sites this is something you might want to look into.

CMUA: PDP-10 — HAMBURG PASCAL
(BILL SCHERLIS)

(1) Some changes in the code were required in order for compilation to succeed here. In particular, the local compiler uses different conventions for `PACK` and `UNPACK` has different switches, and does not want a `PROGRAM` statement. Also, a main program body is not required in a file for separately compiled procedures. These changes were all fairly minor.

(2) The compiler here is not friendly to inter-procedural `GOTO`s, so these were eliminated by adding a new `WrapUp` procedure. (See the labels `endOfTEX` and `FinalEnd` in TEX.) Again, this was straightforward.

(3) Some new features were added to the local compiler (by Andy Hisgen) to support ASCII files and `False-starts`. `FILE OF ASCII` does the expected thing here, except the conventions for RESETing the terminal are somewhat different. `FalseStart` is like the MACLISP SUSPEND operation: If a Pascal program calls `FalseStart`, then execution is suspended and the program may be SAVEd. When this core image is STARTed up, execution will resume at the `FalseStart` call. I added such a call to our copy of TEX.PAS just before the call to `InitSysDep`.

(4) The installation documentation was reasonable, though it could be a bit more detailed in certain areas. Examples: expected problems, the symptoms of various bugs (e.g., not reading the STRINI file), some remarks on the control structure of TEX,...

(5) Testing here has been a bit skimpy, since I can't easily get hardcopy output.

(6) Some hacking still remains: I haven't touched `AppendtoName` yet, but I expect no problems here.

Andy Hisgen suggests changing the procedure `error` so that ordinary letters are used instead of `CR` and `LF`. Thus, the help message becomes something like:

```
Type c or C to continue,
  f or F to flash error messages,
  1 or ... or 9 to dismiss the next 1 to 9
    tokens of input,
  i or I to insert something, x or X to quit.
```

instead of

```
Type <cr> to continue,
  <lf> to flash error messages,
  1 or ... or 9 to dismiss the next 1 to 9
    tokens of input,
  i or I to insert something, x or X to quit.
```

because having a message like this implies that the host operating system will let the user type in both CR and LF and that it will distinguish between them. Some systems do not do this, either because they don't permit it at all, or because it is not the normal way of doing things on that system. Unix, for example, seems to turn both CR and LF into LF. This problem cannot just be smoothed over in SYSDEP.PAS, because the help message above occurs in TEX.PAS and because the procedure error in TEX.PAS is the one which actually fondles the characters to see if we got a CR or LF.

PRINCETON PLASMA PHYSICS LAB:
PDP-10 — HAMBURG PASCAL &
VERSATEC OUTPUT (PHIL ANDREWS)

This is about the first thing I, or anyone else here, have done in Pascal and I had to guess at some of the differences between our compiler and yours.

It seems that TEX assumes that the loader will preset all variables to zero, however our loader inserts junk some of the time.

Since our compiler doesn't have enough room to load in debug with TEX it's particularly painful trying to find errors.

Once I figured out how to bring up the first release I had little trouble with the others but I think some help could be given. The major problem with compiling was the sheer size of TEX.PAS and TEXPRE.PAS which forced changes in our compiler.

As of May 9 I have the latest version of TEX up and running and have no outstanding bugs. Our interface to a 100pt/inch Versatec is working satisfactorily and we are hoping to obtain the use of 200pt/inch Versatec in the near future. I am presently supporting TEX at General Atomic at San Diego also, our spooler only required a slight change to run there.

* * * * * * * * * * *

## TEX AND HYPHENATION

Frank M. Liang

Word hyphenation is a useful feature of any computerized document formatting system. Sometimes it is also one of the most embarrassing.*

The current TEX hyphenation algorithm was developed by Prof. Knuth and myself in the summer of 1977. Our goal was to come up with a reasonably compact algorithm that would find a significant percentage of possible hyphenation points, but would make very few errors. The algorithm is described in Appendix H of the TEX manual. Note that

*If you find any such embarrassing hyphenations done by TEX, you are encouraged to send them to the author.

there have been quite a few minor changes since the original printing of the manual; these are described in the errata file.

Basically, the algorithm has four types of rules: (1) Prefix removal (e.g. `com-`, `dis-`, `ex-`), (2) Suffix removal (e.g. `-able`, `-ful`, `-tion`), (3) Vowel-consonant-consonant-vowel rule (can usually split between the consonants), and (4) Exception table (about 300 entries). Actually, these parts are applied in the order (4), (2), (1), (3); this order is rather important because of the interaction between rules. For example, the `horse-` prefix was put in not so much because we were concerned about hyphenating words like `horse-power` correctly, but rather to avoid hyphenating them incorrectly (the vccv rule (3) would break `hor-sepower`).

The rules were mostly found by hand. Good prefixes were found by looking through a dictionary; suffixes by looking through a reverse dictionary. Other ad hoc rules were discovered as the development proceeded (break vowel-q, break after `ck`). However, as good computer scientists, we then used an on-line copy of the American Heritage Dictionary (at Xerox PARC) to test our rules. This testing had two purposes: (1) to determine which pairs of consonants should be split under the vccv rule, and (2) to generate a list of exceptions to the rules. The exception list originally contained thousands of words, but was pruned down to just a few hundred. Also, in some cases new rules were formulated to take care of large classes of exceptions.

How well does the algorithm work in practice? Quite well, it seems. Quantitatively, in a test on a pocket dictionary word list, the algorithm found about 40% of the allowable hyphen points, with about 1% in error. Furthermore, the hyphen points found were usually the most reasonable or "good" places to break the word. In practice, the algorithm almost never makes a glaring mistake, while at the same time the user does not very often need to specify explicit (discretionary) hyphens, unless the columns are very narrow (or letters very wide).

The algorithm takes about 4K 36-bit words of code, including the exception dictionary.

A note on the implementation: If the algorithm is programmed by sequentially checking each of the rules to see if it applies, it will run rather slowly. Using a hash table would improve things, but a faster and more compact way is to use a version of a finite state machine. Interested readers should look at the actual code.

**Time magazine algorithm**

This is reputedly the most widely used hyphenation algorithm (of acceptable quality). The idea is to decide whether or not to split a word based on four letters `wx-yz` around the potential hyphen point. However, this would require storing a table of $26^4 = 456,976$ bits, which is excessive. (Actually, only about 10–15% of these 4-letter patterns actually occur in English words, but it seems the storage would still be considerable.)

Instead, the algorithm uses three tables of size $26^2 = 676$, corresponding to the pairs `wx`, `xy`, and `yz`. The origin of these tables seems to have been forgotten, but they are supposed to represent the conditional probability of a break given that the first two, middle two, or last two letters, respectively, are a particular pair. To decide whether to break at a given point, the values for the three pairs are looked up, multiplied together (as if they were independent probabilities, which they are not), and then compared to a threshold.

Adjusting the threshold obviously changes the performance of the algorithm. One estimate is: 40% hyphens found with 10% error. In any case a large exception dictionary will be required for good performance. One reason for this is that looking at just four letters around the potential hyphen point is not sufficient. The author has discovered examples where one must look as much as 7 letters ahead (!) to determine hyphenation (consider `def-i-ni-tion` vs. `de-fin-i-tive`).

**Patterns**

Currently the TeX project (more precisely, me) is conducting research into better hyphenation algorithms. In particular, I am investigating a method based on the idea of hyphenating and inhibiting patterns. For example, a hyphenating pattern might be `-tion`, indicating that whenever `tion` occurs in a word, we can hyphenate immediately before it. Another good example is `c-c`. Note that hyphenating patterns are a generalization of the prefix, suffix, and vccv rules discussed above.

In addition, the idea of inhibiting rules has proved useful. Such rules formalize the notion of "we can usually hyphenate such and such a pattern, except when it is followed by ...". Also, such rules are often useful for handling classes of exceptions.

More importantly, we hope to be able to extract the rules automatically from an on-line dictionary. This will be done by collecting statistics on the effectiveness of all possible patterns, and then using some heuristics to choose a good set of patterns. Preliminary experiments with this approach indicate that it will be very effective. For example, a set of about 3300 hyphenating and 2700 inhibiting patterns gets 85% of the hyphens with 2% error.

# TEX on SMALL MACHINES

Kent S. Harris and Robert M. McClure

Unidot, Inc. Sunnyvale, CA

## 1. ABSTRACT

As small computers become more affordable, the demand for increasingly sophisticated software grows. Unfortunately, much of this software is large and does not easily fit on small computers. Emulation through interpretive techniques typically yields unacceptable performance. Reengineering of existing software is frequently desirable, therefore, for reasons of both cost and compatibility. The art of compressing large mainframe-developed programs into small address space machines has become very important. One such effort is the reengineering of the TEX typesetting system. Since this paper really deals with putting large software systems on small computers, and is not a paper on TEX itself, familiarity with TEX is assumed.

## 2. INTRODUCTION

TEX[1] is a recent creation by Donald E. Knuth at Stanford University. It is a system for typesetting beautiful books–especially books that contain a lot of mathematics–an area in which the cost effectiveness of computer typesetting over manual methods is obvious. The equation

$$\frac{1}{2\pi}\int_{-\infty}^{\sqrt{y}}\left(\sum_{k=1}^{n}\sin^2 x_k(t)\right)\bigl(f(t)+g(t)\bigr)\,dt$$

is a vivid example. That TEX is especially popular today compared to other computer typesetting systems is due primarily to TEX's level of sophistication and ease of use. TEX has recently been endorsed by the American Mathmatical Society for submission of machine readable input, a trend that is likely to grow quickly in the years ahead.

The goal of this project was to implement TEX on a small, preferably desktop size, computer. We also wanted to use a commercially available operating system that would provide additional typesetting tools such as editors and other text processing facilities. Since UNIX[2] had demonstrated superior text management functions over other operating systems and appeared as if it were becoming a de-facto standard for small machine operating systems, it was chosen as a basis for this implementation. UNIX was in the process of being ported to several 16-bit architectures by various manufacturers while already enjoying rather widespread usage in the DEC PDP-11[3].

Another issue was that of implementation language. The C language[4] seemed to meet the requirements best for programming this class of system on small processors. It provided the best combination of both high and low-level features of any language that had reasonable wide-spread distribution.

The final major goal was to avoid using compression techniques such as interpretive systems. Although interpretation is widespread today (most BASIC systems are interpretive), and do provide for very compact code, it usually incurs a large performance penalty, especially for programs requiring substantial computation. TEX was expected to be computation intensive.

The original version of TEX was written in SAIL, a language developed at the Stanford Artificial Intelligence Laboratory for DEC PDP-10's and 20's. Furthermore, TEX was a large program, even with 36-bit words. It seemed to have an insatiable appetite for memory while building pages of text. The idea of actually compressing TEX into a 16-bit address space without using interpretive techniques initially appeared quite absurd.

1) The "X" in TEX is actually a Greek chi, and therefore TEX is pronounced the same as the first syllable in technology. The name TEX is a registered trademark of the American Mathematical Society.

2) UNIX is a trademark of Western Electric.

3) DEC and PDP-11 are trademarks of Digital Equipment Corporation.

4) Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey (1978).

The goal indeed seemed to be a formidable task. The methods employed to bring about the realization of "Table-top" TEX is what this paper's all about.

## 3. MULTI-PROCESS APPROACH

It was clear from the size of the SAIL version that TEX could not possibly exist as a single process within a 16-bit address space. Because UNIX does not support a runtime overlay system, we decided to split TEX into two separate but concurrent processes. We decided to use UNIX's simple but elegant system of *pipes*. The UNIX *pipe* is a mechanism by which the output of one process (*pass1*) is routed to the input of another process (*pass2*). One early question concerned whether there existed a division point in TEX where this simple tandem scheme could be implemented, or were feedback paths from *pass2* to *pass1* always necessary?

Figure 1 is a simplified block diagram of data flow. The vertical dashed line shows the most obvious place of division. The balancing of instruction and data space requirements between the two passes along with numerous other details also affected this division point. After examination of the boxes labeled "Main Control" and "State Stacks", it was determined feedback paths would, unfortunately, be required. *Pass1* and *pass2* had to exist as processes coupled by two *pipes*, one for communication in the forward direction and one for the reverse direction.

A few words about virtual memory techniques are in order. As much data (both predefined and generated) as possible was to be kept memory resident for obvious performance reasons. Since a 16-bit data space was simply not enough to hold the numerous tables required by TEX, most were moved to secondary storage and cached through memory resident buffers with buffer replacement done on a least-recently-used (LRU) basis. We will refer to these as virtual memory (VM) systems. How these VM systems are incorporated into various nooks and crannies of TEX will become clear shortly.

## 4. PASS ONE

The overall purpose of *pass1* is to break down input text into a stream of primitives and data for *pass2*, who does the real work. *Pass1* is responsible for macro definition and expansion and for managing other user defined token lists (such as the output routine, alignment texts, and mark texts). These token lists are kept as character strings instead of hash indexes or primitive codes as in the SAIL version. This provides for ease of *pass1* manipulation whereas the SAIL implementation reduces them to *pass2*-like entities. Subsequently, time honored string based algorithms can be used. Figure 2 shows our macro expansion stack frame layout.

The hash entry symbol table is managed on a linear collision basis–applying the hashing function to the symbol, using this value to index into the hash table, and then linearly scanning to find either a symbol match or a free cell. Contents of the hash table are address pointers to the symbol character string (see figures 3 and 4). The hash table is one-to-one with the first part of the equivalents table. The equivalents table contains key information for all primitive control sequences and user defined control sequences. This arrangement is essentially identical to the SAIL version with the exception that the hash table pointers and macro text pointers are actually VM pointers.

The VM system used to hold the output routine, alignment templates, and mark texts is illustrated in figure 5. The addresses shown are not mandantory and can be tailored for a particular user's needs. Additional VM systems can be added to expand allocated sizes of the various elements. However, the numbers shown are realistic. This mechanism of fixed size allocation fosters simplicity and saves memory with little loss of generality.

Two output routine definition areas are shown, allowing output routine redefinition without collision. The next four 4K (byte) blocks comprise the halign and valign template storage areas. The remaining memory up to the 32K point is currently unused. The second 32K is divided into 128 256-byte chunks. Each chunk can hold one *mark text*. Although there are only three possible marks per page of text (top, bottom, and first), *pass1* does not know where the page break will be. To solve this problem, we keep up to 128 *mark texts*. When *pass2* decides it's time to digest the output routine, it indicates to *pass1* how many marks were on the page just built. *Pass1* uses this information to manage the pointers and text buffers accordingly.

## 5. PASS TWO

The primary data structures of *pass2* are large and complex linked lists. For performance, these lists are memory resident. However, two tables are virtual in *pass2*–font information and the hyphenation exception dictionary.

Font information files typically range between 700 and 1000 bytes in length. Since TEX supports 32 font files, the need to keep these tables on secondary storage is clear. As before, font information files are cached on an LRU basis with a small number of files (usually four) in memory at one time. Since font changes are relatively rare, this causes little performance degradation.

In fact, the only VM system that has hindered performance significantly is the hyphenation exception dictionary. Currently, exceptions are kept in a file of sorted fixed length records and simply binary searched. There are superior methods that we expect to incorporate later.

Perhaps the most important implementation decision in *pass2* concerns execution speed rather than code volume. The SAIL version utilizes floating point exclusively for its *glue*[5] values. This approach is unacceptable considering the poor floating point characteristics of most 16-bit processors.

Moreover, output results will differ between various processors with slightly different floating point implementations due to differences in accumulated round-off. The solution, of course, is to use fixed point with appropriate scaling. There is a most conspicuous fly in this ointment. It is inherent in TEX's line-breaking and page-breaking algorithms that *glue* values have the same dynamic range between 0 and 1 as between 1 and $\infty$. This is essentially the definition of floating point!

The technique we chose is usually considered the worst possible solution–software emulation of floating point. The key, however, is the format used (figure 6). At first, a 16-bit exponent may seem a bit excessive, but since this is the natural width of arithmetic of most small machines, it provides for rapid manipulation. Astonishingly enough, TEX has performed very well utilizing this technique.

The SAIL version uses floating point exclusively for all *glue* and dimensional data. We limit the use of floating point to gain both space and speed performance. With the exception of 32-bit *glue* values and line widths used within the line-breaking code, all of TEX's internal dimensional values are kept as 16-bit integers. In this implementation of TEX, internal units are mils (.001 inch), providing for a range of $\pm 32.767$ inches. For a standard 8.5 by 11 inch page, this is certainly suficient.

## 6. CONCLUSION

A few words about performance are in order. Actual measurements so far show that TEX on an Onyx C8002 system can process 8.5 by 11 inch pages of average complexity text in about 5 to 10 seconds per page (with hyphenation turned off). Currently, hyphenation degrades this by a factor of 2 or 3, but this is improvable. The system mentioned has as its processor a Zilog Z8002 16-bit microprocessor with 256KB of main memory, and a secondary store consisting of a 10MB Winchester disk with an average access time of about 55ms.

The techniques described here are only an example of those possible in the realm of software compression. The task of compressing software without gross performance degradation may not be a systematic one, but this example illustrates its feasibility.

## 7. REFERENCES

1. Knuth, D.E., *TEX and METAFONT, New Directions in Typesetting*, Digital Press, American Mathmatical Society (1979).

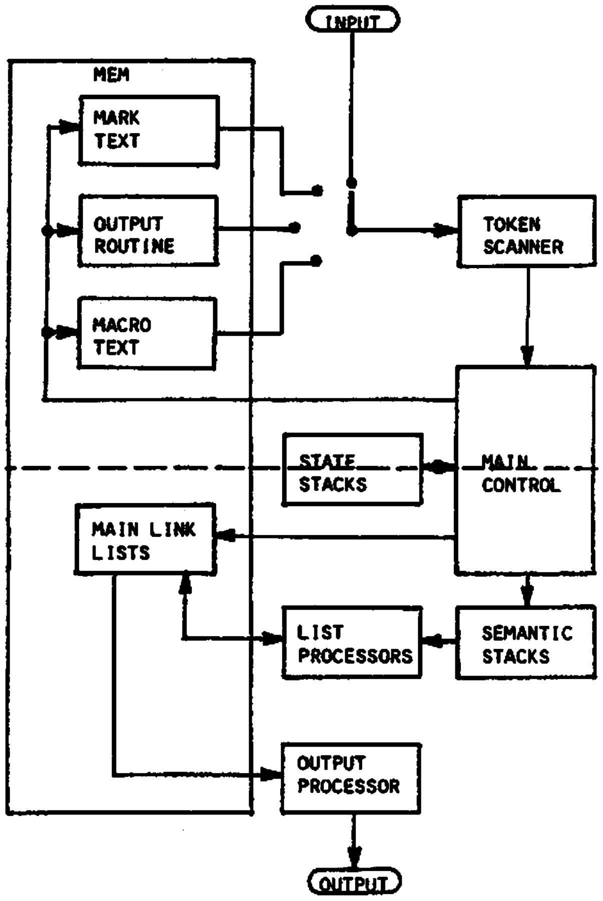


FIG. 1 - SIMPLIFIED DATA FLOW SAIL VERSION

5) One of the more interesting concepts in TEX is the idea that characters and combinations of characters (boxes) are held together with a flexible space called *glue*. After positional calculations are done, the *glue* is *set*.

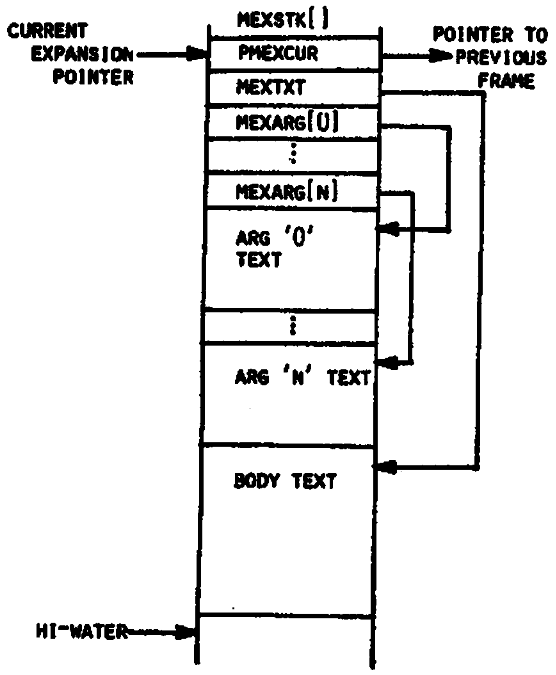


FIG. 2 - MACRO EXPANSION FRAME

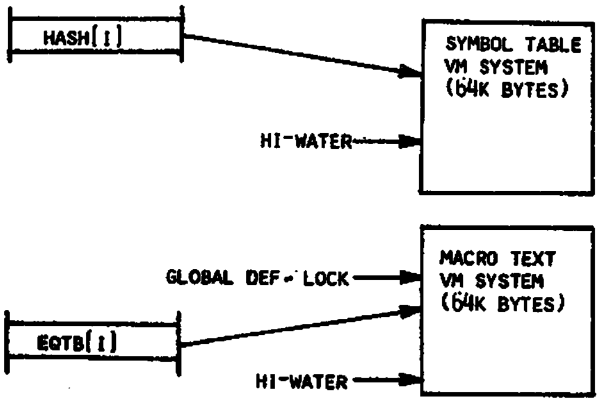

HASH[I]
SYMBOL TABLE VM SYSTEM (64K BYTES)
HI-WATER
GLOBAL DEF. LOCK
MACRO TEXT VM SYSTEM (64K BYTES)
EQTB[I]
HI-WATER

FIG. 4 - MACRO VIRTUALIZATION

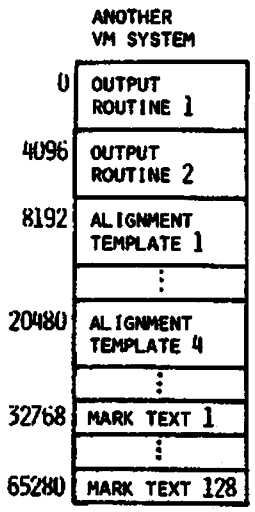


FIG. 5 - MISC. PASS1 VIRTUALIZATION

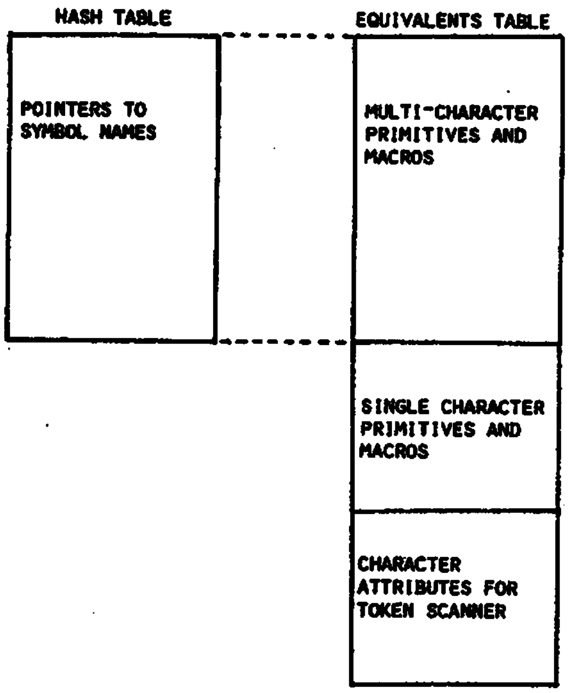

HASH TABLE
POINTERS TO SYMBOL NAMES
EQUIVALENTS TABLE
MULTI-CHARACTER PRIMITIVES AND MACROS
SINGLE CHARACTER PRIMITIVES AND MACROS
CHARACTER ATTRIBUTES FOR TOKEN SCANNER

FIG. 3 - EQUIVALENTS TABLE

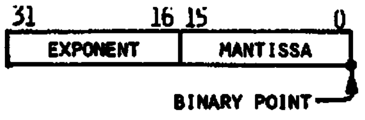


FIG. 6 - GLUE FORMAT

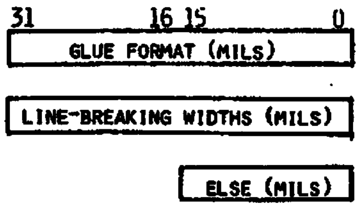


FIG. 7 - DATA FORMATS

* * * * * * * * * * *

## Output Devices

* * * * * * * * * * *

### – OUTPUT DEVICE NEWS FLASH – (APS-5 AND LINOTRON 202)

David Fuchs

A quick note just before the deadline: During this summer, I will be working on Pascal interfaces for the Autologic APS-5 (and the compatible micro-5), and the Mergenthaler Linotron 202. The first incarnation of each interface will allow TEX to use only fonts supplied by the manufacturer. There is some reason to hope, however, that I will also be able to get enough information about font encoding on these machines so that I can write METAFONT output modules for each one. This would allow the CM font family (as well as any other METAFONTed fonts) to be used with these machines. I am particularly enthusiastic about the APS-(micro)5; the native language of the machine looked good, the documentation seemed complete, and the people at Autologic were quite helpful. They even said that they would consider releasing their font encoding description on a non-disclosure basis "if it helps us sell typesetters". For more details on the status of either of these interfaces, please contact me at this address:

David Fuchs
Computer Science Dept.
Stanford University
Stanford, CA 94305

I am also interested in hearing any news of anyone succeeding or failing to interface TEX to any device.

* * * * * * * * * * *

### Summary of Computer Equipment and Output Devices

The following computer architecture groups and output devices have been specified by TUG members. Computers marked with ** are known actually to have a version of TEX installed and capable of producing DVI files; output devices so marked have actually produced output copy from TEX DVI files. (No distinction is made between systems capable of production and those still operating only on a test basis.) A single * indicates that work is in progress.

The separately-bound membership list contains sublistings of member names by the device types given below. There is not yet any cross-reference mechanism to indicate which output devices are connected to which computers; for details, see the individual member listings and the "Site Reports" column which appears in every issue of TUGboat.

#### COMPUTERS

Amdahl
Apple
Burroughs
CDC
CDC 6400
CDC 6600
CDC 6700
CDC 7600
**CDC Cyber
Cray
DEC
**DEC 10
**DEC 20
DEC WPS-8
DG
DG Eclipse
DG Nova
DG S140
Facom 230-75/M-180
Foonly
Fujitsu M190
HP 1000
HP 3000
Honeywell
IBM
**IBM 303X
IBM 3081
IBM 360
**IBM 370
IBM 43XX
IBM Series 1
ICL 1904S
ICL 2960, 2980
Intel 8080
Interdata
Ithaca Intersystems
LSI 11
microcomputers
Motorola 68000
*Multics
**Nord
North Star Horizon
**Onyx
PDP 11
PDP 8
PERQ
Pascal microengine
Perkin Elmer
Prime
Raytheon 703
S-100
Siemens
Singer/Wang L8400
TI 990
TRS 80
Tandem
Telefunken TR440
**Univac 1100
*Univac 90
Univac System/80
VAX
*VAX (UNIX)
**VAX (VMS)
Wang 2200
Wang OIS
Xerox Alto
Xerox Sigma
Z80
Z8080

#### OUTPUT DEVICES

AM Comp/Set 4510
AM CompEdit
Alphakey Multisetter
**Alphatype CRS
Anadex
Autologic APS-5
Bobst Eurocat
CalComp
**Canon LBP
Compugraphic
Compugraphic 7500
**Compugraphic 8600
Compugraphic Editwriter
Compugraphic Unisetter
Compugraphic Videosetter
Diablo
Dicomed D47
**Florida Data
GSI C/A/T
GSI phototypesetter
Graphiset 8
Harris 7400
Harris phototypesetter
Hell Digiset
IBM 3800
IBM 6670
IBM laser printer
III COMp80
III VideoComp
laser
Mergenthaler
Mergenthaler CRTronic
Mergenthaler Linotron
Mergenthaler Omni
Mergenthaler VIP
NEC Spinwriter
Nortext typesetting system
Olivetti
Photon Pacesetter
Printronix
QUME
Sanders
Tektronix
Trilog C-100
**Varian
**Versatec
Wang phototypesetter
Xerox
Xerox 1700
Xerox 5700
Xerox 9700
**Xerox Dover
**Xerox XGP

IMAGEN CORPORATION

page 1 of 2

## Description of IMAGEN products

### INTELLIGENT PRINTER SYSTEM

IMPRINT-10 : A xerographic intelligent printer system developed and introduced by the IMAGEN Corporation, with initial deliveries in May 1981. IMPRINT-10 uses software-definable fonts to print justified text in a variety of styles and sizes at high resolution. Interfaces for the IMPRINT-10 exist which allow it to be easily mated to most host systems. *This page was printed on the system.* IMPRINT-10 is based on the Canon LBP-10 table-top printer which evolved from office copier technology. It prints on plain paper, using a solid state laser and rotating mirror to synthesize images at a rate of about 10 pages per minute for 11 inch high paper, with a resolution of 240 dots per inch.

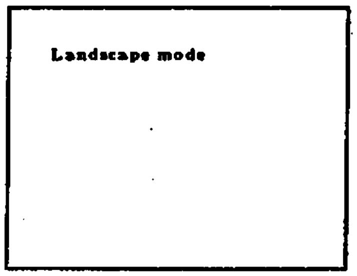
### PRINTER INTELLIGENT CONTROLLER SYSTEM

IMAGEN-C : A general purpose display controller developed and introduced by IMAGEN Corporation, with initial deliveries in May 1981. IMAGEN-C accepts page layout information from a host computer and synthesizes the video needed to control the raster-scanning printer or other display device. It can be interfaced to one or more host computers using a variety of serial and parallel interfaces. IMAGEN-C contains diagnostic procedures for itself and accessible portions of its environment. It is available only to OEM buyers with nationwide maintenance organizations.

### FEATURES AND FUNCTIONS PROVIDED BY SYSTEM INTELLIGENCE

**Type Font Flexibility.** Suitably formatted files may be printed as justified text containing an assortment of type styles, sizes and pitches including proportional. The systems can handle **extended character sets** such as those encountered in ideographic languages (e.g. Kanji symbols: 撒 仔 紙). This permits large symbol sets to be used as well ($\sum_{i=1}^{n} \alpha\psi_i + \frac{1}{\aleph+\rho}$). The commands necessary to invoke font changes are simple character sequences which can be generated by any host system without major software effort. The systems are, however, also capable of printing the output from Troff, TEX, Scribe and similar typesetting systems. Fonts are software-defined and can be local in firmware or downloaded from the host computer to the controller.

**Page Orientation Flexibility.**



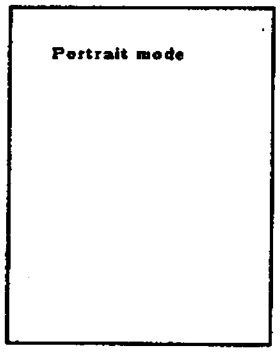


IMAGEN CORPORATION page 2 of 2

**Forms overlay.** Forms can be overlaid with whatever data is sent from the host computer to the printer. A forms definition package allows for forms to be easily described in a high level language.

**Logo and Signature Printing.** IMAGEN has the capability to scan and digitize hard copy originals in advance and store them in the host or locally in the system for printing upon user command. In this manner logos, signatures, as well as other graphic images can be integrated into text output.

**Sorter/Collator.** With this feature, reports may be transmitted from the host computer to the controller with the pages in any order. The controller will then print as many copies of the report as specified keeping all pages in the proper order for each copy set, without the need for further communication.

**Business Graphics.** This feature allows the system to print curves of different shapes and widths as well as backgrounds, intermixed with the text. Business graphics features can be used to produce typical graphs, charts, engineering diagrams, and, in general, drawings of medium complexity.

**Full Graphics Capability.** This feature permits the system to generate an arbitrary pattern of dots, so that any figure may be reproduced on the page. In this way the system can be used as a facsimile receiver, general plotting device, and to do CRT screen dumping for computer graphics systems.

**Text Setting Capability for Word Processors.** In typesetting applications the host computer usually generates the appropriate set of instructions to the controller to obtain the desired layout. Text Setting capability allows for justification and other basic typesetting operations to be performed by the system. This feature can be used to extend the capabilities of existing word processors, most of which operate only in a "typewriter" style.

**Communication Protocols.** The systems have the capability to communicate using network protocols. This feature allows the easy integration of IMPRINT-10 as a node within a distributed environment.

**File Management Capability.** An optional local disk and file management capability allow for font storage and printing job queueing. This results in a **self-contained printing system** in the sense that the host computer does not have to undertake font management and spooling tasks.

**BASIC SYSTEM.**

- Four fonts
- Sorter/collator
- Page orientation flexibility
- RS-232C interface

By selecting a suitable combination of features and an interfacing arrangement, this versatile printing system can be adapted to a wide range of tasks for word processing, business data processing, communications, in-plant printing, and graphics.

IMAGEN *12769 Dianne Drive, Los Altos Hills, CA 94022* 415 949-1580

* * * * * * * * * * * *

## Site Reports

* * * * * * * * * * * *

### TEX UNDER THE NORTH STAR

#### Part I

As coordinator for the CDC Cyber implementation of TEX, I have agreed to answer questions from interested Cyber sites about our progress with TEX-in-Pascal and with our implementation of drivers for the output devices to which we have access. When we have succeeded in our implementation, I will make available for distribution, for a charge, a tape with the following information:

1. TEX-in-Pascal source, as modified to compile successfully on a Cyber under the version of Pascal maintained at the University of Minnesota.
2. A file containing the Computer Modern font set.
3. Our device driver(s).
4. Relocatable binary of these files.

I will also send a copy on microfiche of the Pascal compilation and cross reference listing. I will provide sympathy, understanding, and handholding as you put TEX up on your system.

I have also agreed to act as go-between for other matters which may come up from time to time. However, I don't intend to act as a roadblock! Bob Welland, editor of TUGboat, would be delighted to receive articles and letters-to-the-editor directly from you.

My address is in the front of TUGboat; my phone number is (612) 373-4599. This number is always answered and I am reasonably good about returning calls.

In Part II of this report, M. J. Frisch, our TEX implementor, describes our current progress. However, please note that calls should be directed to me, not to him.

Thea D. Hodge

#### Part II

We recently got a proper device independent (DVI) file output from our Pascal version of TEX. This is a version we received from Stanford in December, 1980. While we are pleased with this success, it is only a beginning.

Several temporary changes had to be made to get TEX to work because our conversion is incomplete. Our font information file had only one font which was converted to 60-bit Cyber words from 36-bit DEC-10 words (containing mixed integer and floating point). The input text was very simple and the output was only a set of numbers, the contents of TEX's DVI file.

We have no device driver programs and our next major task is to write one or more. Potential devices at our site are a Varian 200 dot/inch plotter, a Linotron 202 typesetter, plus a Xerox 9700 page printer at a local service bureau. (The 9700 would be intended for proof copy of text; full math output doesn't seem practical on it at present.)

Further tasks are to finish the TEX conversion by finding out how to remove the temporary changes. We believe these changes are due to incomplete implementation of the system dependent routines. We will also have to convert the remaining font information files to Cyber format.

Our system dependent changes produce a NOS operating system version using 6- and 12-bit ASCII character codes for input and output. If we can find a way to test it, we can build an 8-bits-in-12 ASCII version for NOS/BE sites.

TEX is a very large program, around 98K words. Our site only allows time-sharing for 32K so we run TEX as a batch program. While we realize that TEX is intended to be interactive, our CDC Cyber users will have to live with this problem until (or if) we can find ways to make our version of TEX smaller.

If all goes well, we plan to announce TEX availability for CDC Cyber users at the VIM users group meeting in October. We will need to write installation documentation and we would like to try out our installation procedure at a nearby CDC site.

If there is enough time before October, we will do a conversion of a more recent version of TEX (rather than the December 1980 version). There are a number of attractive features in recent versions which would be helpful. Several recently written Stanford programs are also valuable, particularly one by David Fuchs that prints the contents of a DVI file.

In summary, we really can see some light at the end of the tunnel.

Michael J. Frisch

* * * * * * * * * * * *

### DECSystem-10/20 IMPLEMENTATION WORKSHOP

Phil Sherrod

An implementation workshop for persons interested in installing TEX on a DECSystem 10 or 20 will be held at Vanderbilt University in Nashville, Tennessee, on September 10 and 11. The course will discuss the details of connecting TEX output devices to DEC 10s and 20s as well as software issues involved in installing TEX. Conference attendees will have access to a system running TEX. For details, contact

Phil Sherrod
Vanderbilt University
Box 1577, Station B
Nashville, TN 37235
615-322-2951

* * * * * * * * * * *

## AMS SITE REPORT

Barbara Beeton

Since our last report (TUGboat Vol. 2, No. 1, page 48), an integrated spooling system has been developed and installed to control output to both the Varian and the Florida Data. This spooler will print straight text (ASCII) files, using the rather primitive fonts supplied with the output devices, as well as TEX output. Output defaults to the "local" device, using device-appropriate fonts (Varian in Providence, Florida Data in Ann Arbor), but may be overridden; output page positioning may be adjusted either interactively or from a predefined option (.OPT) file, allowing each column of a multicolumn page to be treated by TEX as a separate page to conserve memory space, then overlaid on output; multiple copies, output of selected pages, deletion (by submitter) of a job from the queue, and other similar features are also supported.

We have acquired the capability of generating fonts for the Alphatype via **METAFONT** (see articles by W. J. LeVeque, page 39, and Ron Whitney, page 40). The Alphatype is being used for limited internal production. (Another composition system is still being used for our major journals, pending completion of necessary work on symbol and math fonts.) The entire two-column portion of this issue of TUGboat and the separate address list have been generated on the Alphatype. A number of articles were sent in on mag tape, from VAX and Univac 1100 systems as well as from DEC 10s and 20s; this experiment has been most successful, and we hope to receive many articles for future issues on tape. Some files have also been received via a phone file-transfer mechanism (not a network).

We have prepared camera copy on the Alphatype for several other organizations that do not have their own high-quality output device; for these jobs, TEX input was provided to us on tape, and the output proved to be virtually identical to the original. Requests for Alphatype jobs will be considered, on a time-available basis; for price and other details, call or write to

Raymond Goucher
American Mathematical Society
P.O. Box 6248
Providence, RI 02940
401-272-9500

* * * * * * * * * * *

## TEX AT THE 1981 SPRING DECUS U. S. SYMPOSIUM

Patrick Milligan
BNR Inc.

In keeping with a long standing tradition, there was a TEX Birds-of-a-Feather session at the recent DECUS symposium held May 18–21 in Miami. (In case you didn't know, DECUS stands for "Digital Equipment Computer Users Society.") The panel members were:

| | |
|---|---|
| Patrick Milligan | Coordinator for DEC-20 |
| Phil Sherrod | Coordinator for DEC-10 |
| Barry Doherty | American Mathematical Society |
| Bob Phillips | Oregon Software |

It comes as no great surprise that DEC users are interested in TEX: Don Knuth's book, *TEX and METAFONT: New Directions in Typesetting* is jointly published by the American Mathematical Society and Digital Press. In addition, versions of TEX exist for the DEC-10, DEC-20, and VAX/VMS computers.

Since the DECUS Symposium was held immediately after the TEX Implementors' Workshop at Stanford, emphasis was placed on the installation of TEX. Output samples from various devices (including Versatec, XGP, Canon LBP-10, and Alphatype) were passed around. A free TEX sample (the TUG membership form) was distributed. The content and utility of recent TUGboat issues was described. In short, a sales pitch was made for TEX, TUG, and TUGboat.

Another session on TEX is planned for the next DECUS Symposium to be held December 7–11 in Los Angeles.

* * * * * * * * * * *

## TEX AT NIH

Rachel Schwab

TEX exists on the DECsystem-10 at the National Institutes of Health in Bethesda, Maryland. At present it may be used only by the DECsystem-10 staff. Hopefully we will soon have facilities available so that it may be opened up for general usage.

Currently there are four steps involved in running TEX at NIH. First one runs TEX to translate text into a DVI file. Then the DVI file is used as input to a program called DVIPDP. DVIPDP essentially takes DVI records, extracts all the important information, and outputs new records that contain opcodes along with other information. For example: opcode 3 indicates a rule and the information that goes with a rule are height and width. Opcode 2 indicates a "delta y" item—the vertical coordinate is moved by a certain number of pixels, et cetera.

DVIPDP produces a "PDP" file of records like the ones described above. This file is transferred to a PDP 11/70 by means of a high-speed communications link. A Benson/Varian printer plotter is hooked up to this PDP 11. A driver program, TEX/11, was written for the 11 that takes "PDP" records and translates them into scan lines.

TEX at NIH has the ability to take graphs and figures and place them inside a TEX file. TEX itself has not been modified to do this, but the DVIPDP and TEX/11 driver have the ability. Graphs are created by using MLAB or OMNIGRAPH on the DECsystem-10. A program has been written that will convert a plot file into a file of alphanumeric records which in turn may be translated by DVIPDP.

As one can see by the above description, using TEX at NIH is complex. The four steps (TEX, DVIPDP, CLINK (transfer to the 11), and TEX/11) will need to be compressed before TEX can come up in a production environment. This necessarily involves getting our own small computer and printer to hook up to the DECsystem-10 for usage with TEX.

NIH is a community of scientists and researchers who produce technical papers in abundance. The potential for use of TEX at NIH could be very high. However, TEX is not an easy system to learn. We are hoping that the AMS-TEX macros will make using TEX a little easier, and we are looking into the possibility of writing some type of TEX preprocessor.

* * * * * * * * * * *

## AN IMPLEMENTATION REPORT FOR THE UNIVAC 1100

Bill Kelly
University of Wisconsin-Madison

We have TEX running on the Univac 1100 using University of Wisconsin-Pascal. This is a report on some of the problems we encountered in implementing TEX. The difficulties were of several kinds. A major difficulty is in the differing syntax of external compilation between various Pascal compilers, a feature not defined in the original Pascal definition. Differences in I/O on the host machines and in the ways the compilers handle I/O were another problem. Memory limitation problems were encountered, and these were made worse by the fact that we were altering Pascal code rather than the macro language TEX was written in. The typesetter we are using is not addressable as a raster device, and so we were unable to use standard TEX/**METAFONT** fonts. Instead we had to write a pre-processor to convert descriptions of the typesetter fonts into TEX format. Various errors occurred in using TEX, some of which were traced to errors in the TEX code, others of which related to improperly formatted fonts. The 1100 operating system allows a user to end terminal interaction with a program by typing an end-of-file signal—TEX doesn't consider the existence of this concept. Because we have no inexpensive proofing device we had to write a line printer proofing program which has severe limitations due to the limited range of actions and characters possible on a line printer. We have found the overfull box messages to be unhelpful in correcting justification problems and have replaced them with a more informative message. We have found it very difficult to set type in narrow columns with TEX. Details of all these problems are discussed below.

We should all hope that the project of transporting TEX makes future language designers careful to be complete in defining language standards. Hours and days of work could have been saved if the Pascal definition included external compilation, the default case in the CASE statement, etc.

External compilation in UW-Pascal uses a syntax very different from that used at Stanford. Any shared types, variables, procedures, and functions must be declared in a separate environment module. Only the head of procedures and functions is included there: the name, parameters, and result type comprise the head. Each code module includes a list of procedures and functions which correspond to the heads in the environment. To convert TEX into this format required a lot of hand coding.

Differences in I/O handling required a lot of recoding in the system-dependent module, SYSDEP. Of course, it is to be expected that one would have to recode the system-dependent code, but it could be made clearer what part of the Stanford code refers to PDP-10 peculiarities and what is required by TEX. For instance, TEX expects lines to be ended by carriage returns or line feeds, etc., which is the internal file format of most PDP computers. In the Univac file architecture, carriage returns and so forth are not preserved in the file. This must be the case with many computer systems, and especially in Pascal compilers where such system-specific things are supposed to be transparent. What we did was to read a line from the Univac file, and when Pascal returned true as the value of the end-of-line function *EOLN*, SYSDEP appended a carriage return. This would have been much easier had the SYSDEP code not also dealt with ignoring line feeds and nulls, or rather had it documented and localized these local quirks. To make it easier for future sites to implement TEX, procedures like SkipTrailingStuff to ignore nulls after a line end, and functions like EndOfLine which would check for PDP-10 line end characters, but note that on non-PDP machines a function like *EOLN* might be used.

The fact that TEX expects input files to be page-oriented is a minor nuisance. Apparently files on the PDP-10 are made up of "pages" delimited by a form feed character. This is not the case on our machine and on many others. This causes minor problems in two respects: one is that if a file containing a form-feed character on our system were read by TEX, it would be interpreted in a manner inconsistent with our operating system. The other is that "page numbers" appear along with line numbers in the error messages. This tends to be very confusing since there is no page number in the file structure, but there are page numbers in the formatted output—so a Univac TEX user would assume the page number referred to the output pages. We circumvented this by changing the error message from "p.0, 1.50" to "Line 50". This could be built into TEX by providing a Pascal constant "pageOriented" which the TEX implementor could set to true or false.

Implementing TEX is difficult because the program is written in a macro language which is converted by the UNDOC program into Pascal. Unfortunately the UNDOC program is not presently available in Pascal insofar as we know. The macro code serves well as structured documentation, but it is difficult to refer back and forth between the macro code and the Pascal code. This is because the macro code is divided into sub-sections in a structured heirarchy, which expand into linear Pascal code. Also the use of macros to provide alternate names for variables in the macro language means that a single Pascal statement may look entirely different in the two listings. Another drawback is that expressions involving constants are folded into a single constant. This makes compilation efficient but changing array sizes difficult. We ran into this problem in trying to reduce the size of the "mem" array to meet a memory restriction in our compiler. The ultimate solution is to distribute UNDOC with TEX. But admittedly this work should take second priority to work on TEX itself. A difficulty which may be impossible to surmount with this approach is that the output from UNDOC may still be unsuitable for compilation, depending on the features of the compiler. For us this came about because of the environment feature of UW-Pascal; other sites have had to hand code the default case of the Pascal CASE statement. If this situation occurs, it would be impractical to correct errors in the macro source and all fixes will end up being done in the Pascal code anyway. It is unclear whether it is realistic to expect UNDOC to be able to produce compilable code for all compilers automatically.

On the Univac 1100, a user at a terminal can type `@eof` to force an end-of-file condition in the terminal input to the current program. The program must terminate without further intervention. This possibility is not allowed for in TEX, most likely because this concept of a terminal end-of-file does not exist on many computers. We have circumvented this problem by having the InLnTer procedure in SYSDEP pretend that the user typed `\end` if he indeed types an `@eof`. This is not the ultimate solution, since there are circumstances where TEX will not accept a `\end` command, as for instance in the middle of a paragraph. To help out with this problem (and to make TEX generally more friendly), our basic macro file redefines the `\end` command to be `\par\vfill\end`. This avoids the problem of an unfinished paragraph at the end, but not of an unfinished `\halign`, etc. The suggested solution to this is to have a function TerEof in SYSDEP to detect a terminal end-of-file condition. On detecting this, a clean-up procedure would terminate any current levels of processing, possibly issuing warning messages. Without having examined the code closely, we don't know if the TerEof idea would be the best approach to modifying the existing TEX code, but the general idea would be the same no matter what the implementation. This added code would not impact sites that do not have a terminal eof concept, because the TEX installation documentation would say to have TerEof always return false at these sites.

## Using TEX with a non-raster device

Our primary output device is a Compugraphic 8600 typesetter. The 8600 does use digitized data to create its characters, but it will not accept user-defined character shapes. Therefore our TEX font files reflect the shapes of font characters available from Compugraphic, and our device driver outputs characters and commands to the 8600 rather than bit rasters. We get an excellent quality of output this way, but the approach is not without its difficulties.

One problem was in converting character shapes into TeX-readable format. The data had to be manually typed, and a program was written to transform the raw data into a pair of files, one for TeX and one for the 8600 driver. Until all the bugs were worked out of this program, mysterious TeX errors resulted from improperly formatted font files. For example, parentheses (and other delimiters) of varying size form a linked list in the font information. At first, our font preprocessor did not correctly indicate the end of the linked list. This problem was difficult to diagnose, because it caused a Pascal error; TeX assumed the font information to be correct and so did not issue any warnings. This is as it should be for maximum efficiency of TeX, but it points up the usefulness of a program such as the one described in TUGboat Vol. 2, No. 1 which automatically (and correctly!) converts font files into readable data and vice versa.

Using a device like the 8600 limits the portability of our TeX-produced documents because the fonts do not correspond exactly to **METAFONT** fonts. We have similar fonts, for instance our English Times resembles Computer Modern Roman, but slightly different widths can lead to different paragraphing and pagination, overfull entries in tables, etc. We cannot use **METAFONT** with the 8600 because the manufacturer provides no method of addressing the typesetter as a graphics device, and the cost of having fonts custom-made by Compugraphic is prohibitive.

An interesting problem encountered in using a micro-computer driven typesetter like the 8600 in conjuction with TeX is the lack of cooperation between the two devices. The typesetter itself has a fair degree of intelligence, but it seems impossible to use any of it with TeX because of the need to make the DVI file device-independent: TeX must assume it is dealing with an entirely stupid device! The 8600 does justification, tabulation, automatic accent placement, box-drawing, etc. The commands we pass to the typesetter bypass all these features! The question of cooperation between smart devices is a difficult one, and it seems a shame that the "smarts" of the typesetter can't be better utilized. This is a general question, not one addressed specifically at TeX, that of portability vs. efficiency for a particular device. One approach would be to have a heirarchy of commands, for instance a 'box' command in TeX, which could also be expressed in terms of more primitive commands like HORZRULE and VERTRULE. A user with an intelligent device could configure TeX to output the 'box' command while other configurations could output 4 separate line commands. To maintain transportability, a standard skeleton for a device driver would have to be distributed which could break a box command down into 4 line commands at that stage. The user with the intelligent machine would bypass this procedure. Alternatively, all configurations of a program like TeX could output the box command and distribute the rest of the work to be divided between the device driver program and the device as the programmer sees fit.

**A Limitation of TeX**

We have found one apparent limitation to TeX: it seems impossible to set text in narrow columns without extensive manual hyphenation. One customer wanted to set 9 point type in 2 inch columns. When we ran TeX, we got many overfull box messages. This leaves you to figure out the cause of the error. It may well be that there is a word that TeX does not know how to hyphenate. We have developed a non-standard message for overfull horizontal boxes that prints the box as a single line on the terminal, representing characters as their corresponding ascii character (or a question mark if the character is non-printing), glue as a space, and hyphenation nodes as dashes. This gives you an idea of whether TeX knows how to hyphenate a word at the beginning or end of the line, and you can insert discretionary hyphens as needed. However, if the word that is giving TeX trouble occurs on the previous or next line, you must consult TeX's hyphenation rules to find whether it knows how to hyphenate the word. Thus I have found myself hyphenating words at random, which is a distinct inconvenience in itself, and often find that there are still overfull boxes.

In addition to poor diagnostics, we had trouble in getting TeX to set the narrow columns at all without overfull boxes. We tried adjusting the `\jpar` parameter to have TeX look at more possibilities, and it still gave overfull boxes. Manual hyphenation did not work. We eventually had to increase the `\spaceskip` (or word spacing) parameter to allow a very large space between words, and create typographically bad output. In some cases even this did not work, and a `\linebreak` command had to be used.

At MACC we charge customers as closely as possible for the resources they use, and we have found TeX to be very expensive in both memory and CPU time. This is not really a surprise to anyone, of course. But it means that most users cannot afford to run a document through TEX four or five times to find the right discretionary hyphens needed to avoid an overfull box. Nor is this user-friendly.

There may be no easy way to solve this justification problem. It may be an unavoidable side effect of the justification algorithm used. One possibility would be having a "paragraph debugger" that would help to diagnose the problem in a given paragraph: e.g., point out a large word that cannot be hyphenated that is at the root of the problem. This would be a difficult program to write. An alternate hyphenation routine would be another possibility. Allowing TEX to violate its hyphenation rules if necessary to justify a paragraph, meantime issuing a warning, would at least focus on a specific problem, i.e. a questionable hyphenation. rather than leaving the user guessing.

**Usefulness of macro sharing**

TEX in itself seems to provide a standardized language for typesetting. However, a non-computer programmer will find this language difficult to use, and will not have a ready facility for writing his own macros. TEX users need to do much more macro-sharing as was done in TUGboat Vol. 2, No. 1. Good macros put the typesetting capability in the reach of most users. There are still problems when an error occurs through improper use of a macro. The diagnostics are geared toward one with a knowledge of programming. One aid in this would be to provide a way for a macro to send a message directly to the terminal. This was shown in TUGboat Vol. 2, No. 1 as an extension added by one site; it seems to be a useful enough feature to incorporate into the standard TEX.

* * * * * * * * * * *

## AVAILABILITY OF OREGON SOFTWARE IMPLEMENTATION OF TEX FOR THE VAX/VMS

Monte C. Nichols

This VAX/TEX site report consists of an abbreviated version of a memo recently sent to all VAX/TEX users. The information contained here should allow anyone who obtains the VMS version from Oregon Software to implement same on their VAX/VMS system. The only output device supported at this time is a Versatec 1200 A or V80 printer having a DMA interface to the VAX.

A preliminary VAX/VMS implementation of TEX, with auxiliary programs to support a Versatec printer, is now available from Oregon Software. (*Further progress on a* UNIX/*TEX at Brown awaits the arrival of the U. of Washington compiler.*) The accompanying article by Barry Smith explains how you can obtain a copy of the VMS version. You should understand that, in spite of the efforts made by Oregon Software, this is not the final version (in fact, Stanford has just released a new version of TEX). Barry has overcome numerous bugs in DEC Pascal to get us to our present state, but more remains to be done. You can help in this effort by carefully documenting any bugs that you encounter after making sure (insofar as possible) that they are bugs and not errors you are making during your TEX learning process. We have encountered our share of both in the short while the system has been up and running at Sandia. If you fix a bug, please send your fix to Barry!

The distribution tape is 9 track at 1600 bpi and consists of about 6500 blocks. The tape was made using the command `MCR BCK MTA0:TEX.BCK=*.*;*`. The data can be recovered by creating a directory "`[TEX]`", setting your default to that directory, mounting the tape (using `MOU/OVER=ID MTA0:`), and then recovering the files using

```
MCR RST *.*;* = MTA0:TEX.bck
```

It is suggested that you put all the files on the tape in the [TEX] directory and initially use the program there.

The special Versatec driver (`LVDRIVER.EXE`) must be copied from [TEX] to [SYSEXE] and installed. This is done for Versatecs utilizing a DMA interface by inserting two lines in `[SYSEXE]STARTUP.COM` after `$ RUN SYS$SYSTEM:SYSGEN` but before `AUTOCONFIGURE ALL`. The two lines to add are

```
LOAD LVDRIVER/DRIVER=SYS$SYSTEM:LVDRIVER.EXE
```

and

```
CONNECT LPA0/ADAP=3/NUMVEC=2/VEC=%0174
  /CSR=%0777510/DRIVE=LVDRIVER
```

Reboot the system after these changes have been made.

The programs `LVSPOOL` and `LHSPOOL` spool output from the DVI file to the Versatec printer, placing pages of TEX output vertically or horizontally on the printer. To function properly both programs need the privileges `ALLSPOOL` and `PHY__IO`. When running from the `TEX` account the user must have these privileges; if `LVSPOOL` and `LHSPOOL` are run in `[SYSEXE]`, the system manager can install them with these privileges so they can be used by any unprivileged user.

Although you could start your first *TEXperience* by using TEX interactively, or with a file built using your system editor (using Knuth's *TEX and* **METAFONT** as a guide), it is suggested that you might want your first attempt to be with the file TYPO.TEX. To run this example from the [TEX] directory (without using the startup command file provided) enter:

```
R TEX (Wait for the * prompt—
       this may take up to 20 seconds)
\input typo (Note lowercase.)
Watch the TEX comments—wait for system $ )
R LVSPOOL or R LHSPOOL
     (the system should handle the rest)
```

The mailing list for future informational memos regarding VAX/TEX will be the list of **DUES PAYING** TUG members who indicate a VAX interest, so make sure you join.

* * * * * * * * * * *

## TEX FOR VAX/VMS

Barry Smith
Oregon Software

Well, it works—TEX for the VAX running VMS is alive, available, and in production use. (Production use is defined by example—we've just finished a 192 page manual for our optimizing PDP-11 Pascal compiler that is entirely typeset by TEX, including charts and diagrams.)

It's not yet perfect, nor something a "naive user" should be expected to enjoy. To list the major deficiencies:

- There are still annoying bugs, such as the flaming crash that occurs when one tries to insert a footnote. Most of these seem to be due to bugs in Digital's VAX Pascal, which just makes them harder to trace. (There's a new version due out soon, as always.)
- It's the "old" TEX-in-Pascal acquired from Stanford in late December. (Some of the bugs are real TEX bugs, too.) We've just received the recent official release, and as soon as I get a free weekend, ...
- There's absolutely no documentation whatsoever that relates to the VMS version. (Garçon, another weekend, please!) The amazing thing is that this doesn't seem to matter—Knuth's book describes the input format *exactly*, down to subtleties like tracing, and the installation directions are rather concise: "put everything in account [TEX]".

But, when it's good, it's very, very good—we're converting our skeptics to TEXnicians.

Details: TEX-in-Pascal for the VAX (11/750 and 11/780) running VMS, with about 50 Computer Modern fonts in assorted sizes. Comes with two spoolers (horizontal and vertical) for a Versatec 1200 printer/plotter, using the standard Versatec interface. Uses about 7000 blocks of disk space (mostly for fonts) and about a megabyte (whew!) of virtual memory while running. Comes in source and binary/executable forms, sources for spoolers, utilities, etc. (i.e., we'll send you everything we have). If you want to play with the Pascal programs, you'll want the fancy listings available from Stanford. A copy of the Pascal manual mentioned above will be included on request.

Supplied only on magnetic tape (600 foot) in "BCK/RST" format (that's an MCR utility for backup/restore, which works well for binary files). We can write 800 or 1600 b/in tapes—hearing nothing, we'll send 1600.

Fees: fifty dollars will get you a tape and shipment via UPS. To minimize our overhead, please don't send a tape, and do send a check or negotiable securities so we don't have to deal with purchase orders and billing (I'd prefer small unmarked bills). If you (or your friend) are in the "truly needy", just let me know.

Maintenance: We'll be working on bugs and convenience improvements for the perpetual future, and will be pleased to hear comments, suggestions, gripes, bugs—no promises of any response except through the TUGboat and Monte's newsletters.

* * * * * * * * * * *

# Fonts

* * * * * * * * * * *

## FONT COMMITTEE

Barry C. W. Doherty

Getting TEX up and running is only the first step in producing output. One needs also an output device with an appropriate driver and fonts. Transportable TEX output requires compatible fonts. Those sites which have similiar output devices (Varians etc.) and are willing to use the Stanford Computer Modern family have this problem solved.

There are many, however, who feel limited by the CM family as available from Stanford, either for reasons of design, completeness or merely preference for more traditional fonts (Helvetica, for instance).

In addition, many typesetters come with their own font libraries which people would like to use. A strong interest in these matters was shown by nearly everyone attending the TEX Implementors' Workshop. One result was the formation of a committee to investigate font-related issues. Members of the committee are

| | |
|---|---|
| George Ball | Washington State University |
| Barry Doherty | American Mathematical Society |
| David Fuchs | Stanford University |
| Tom Hickey | OCLC |
| Ron Whitney | American Mathematical Society |

It has been suggested that this committee should first try to identify vendors who are willing to work with the TEX community to provide font information (for TEX's metric files) and digitized representations of fonts (so that these fonts can be used with proof-quality devices), and to provide additional symbols for use with TEX. The role of vendors is complicated by licensing agreements, royalties, etc., and inter-vendor cooperation is unlikely. However, the TEX community may be large and strong enough to have the potential for affecting this development positively and to its benefit. It is probable, however, that such help from vendors will not produce fonts which can be reproduced on many other devices.

The need for a font library has also been pointed out, either one maintained centrally or with some central means for pointing people in the right direction. Many tools are already available for manipulating font files (e.g. translating them into editable form and repacking them). More such programs are needed, and they should be as transportable as TEX. (This could also help with the problem of disseminating font information, avoiding binary data entirely.) There is also a need for "device-independent" fonts, at least to the extent of uniform font metric files and a reasonable representation for proofing, so that a TEX file from an institution that had, say, Helvetica could have that file printed elsewhere.

Another role for the committee might be as a source for model RFP's to aid those institutions which must submit such documents to vendors; this would simplify the task of acquiring printers and fonts.

We welcome any ideas or comments.

* * * * * * * * * * *

## THE STATUS OF METAFONT AT OCLC†

Thomas B. Hickey

We have been running METAFONT on Tandem hardware since August 1980. This was accomplished by recoding the SAIL version into T/TAL, the systems programming language on Tandem. The conversion took about six to eight weeks of concentrated effort plus another two to three weeks over nine months to add additional output modules and correct problems.

The system now is a full version of METAFONT, compatible with the original except in the following respects:

- TFM files are not yet generated
- All of an identifier's name contributes to its uniqueness, not just the first five characters plus length
- The limit on the size of **wxy** subscripts was lifted
- Use of <>, >= and <= for SAIL's one character relational symbols
- Limitations on the size of the raster (currently 300 × 300)
- Output routines adapted to drive Anadex printers

The conversion went fairly smoothly, the major problems being the lack of SAIL's nice strings and the difficulty in determining exactly what field in a word the SAIL version was using and what implicit initializations were being performed by shifts used to access and load fields.

### Experience with METAFONT

In general, working with METAFONT is a pleasure. The interpretive nature of its execution allows a number of powerful tracing mechanisms which are very useful. In designing an alphabet the most difficult problems are deciding on the basic approach to be taken in specifying characters and writing the base routines which support such an approach. Once this is established, adding characters is fairly straightforward.

Most of the problems encountered in writing METAFONT programs arise from the complexities of the characters themselves, not METAFONT, but there are several features which users of other programming languages will occasionally miss:

- Loops
- Combinations of Boolean expressions
- A full complement of intrinsic functions
- Ability to write functions

†OCLC (Online Computer Library Center) is a not-for-profit computer library service and research organisation based in Dublin, Ohio.

- Local variable hiding

Implicit in the design of METAFONT is the assumption that users have the ability to add their own output modules for specific devices. The METAFONT language offers no read access to the raster, so such routines must be written in the implementation language to be included in your METAFONT system. It is also possible, of course, to write programs which manipulate files in existing output modes, such as CHR, but this would not be a METAFONT program.

In my work in alphabet design I have found that much of the code is essentially defining two paths and filling in between them. Successors to METAFONT will undoubtedly include the ability to define and manipulate paths as entities. The concept of pens and erasers is certainly useful, but less fundamental than paths.

**Conversion Problems**

One of the reasons TAL was chosen for implementing METAFONT is its closeness in many ways to SAIL, but there were a number of problems:

- Lack of labelled case statements:
  This was overcome by setting up arrays of addresses for jump tables.
- A limit of two to nesting of procedures:
  Only a problem in the raster module. Overcome by moving and renaming variables.
- Lack of SAIL's strings:
  This necessitated the writing of a fairly comprehensive set of string handling routines.
- Lack of a macro facility as powerful as SAIL's:
  Some sort of macro facility would seem to be needed for a clean translation and TAL's was adequate for most purposes.
- Restrictions on file names:
  File names on the Tandem are limited to eight alphanumeric characters. Output file names are constructed by adding a single character to the front of them, so that CMR10 could produce files CCMR10, PCMR10, TCMR10, etc.
- 16 bit words:
  This probably had the greatest effect of all. The Tandem has an excellent repertoire of 32 bit arithmetic instructions but shifts and Boolean operations are only performed on 16 bit words. For ease of implementation and run time efficiency a 3 word 48 bit cell was used for dynamic memory to correspond with SAIL's 36 bit word. This does waste some space, especially in the double cell section (VMOM) of dynamic memory, but is quite efficient since shifts and masks are not required to access fields. In addition no new restrictions on sizes of fields in memory cells were needed.
- Different rounding of negative floating point numbers:
  Rounding is now performed by a function call.

**Chel**

I am presently undertaking the coding of an alphabet called Chel. This alphabet is based primarily on Helvetica and Helios and is designed to be usable over a broad range of widths and boldness. Chel is set up so that for each font to be generated a call to `Chelbegin` is made specifying point size, boldness and width. Boldness can have any of six values (extra light, light, regular, medium, bold, extra bold) and three width values (condensed, regular, extended). The `Chelbegin` routine then sets up the pens, x height, and other parameters which are used by the character routines. None of the routines assume that the vertical and horizontal raster resolution are equal.

The most important parameters are `efactor` and `bfactor` which control the expansion and boldness. If a user is not satisfied with the range of values generated by the standard widths and boldness these parameters can be controlled directly. The characters are designed so that in nearly all cases they change smoothly with `efactor` and `bfactor`. Exceptions to this include the tail on the lowercase 'a' which has an abrupt transition when the boldness is increased. I estimate that designing characters to work over such a broad range takes several times the effort that designing a single font would. Individual characters can be completed in as little as 15 minutes, or may take several days. An average character takes 3–4 hours.

Rough routines for both upper and lower case have been completed, but a great deal of work remains to be done to refine them and make them work correctly in all weights and widths. Figure 1 is a lower case 'b' in proofmode. Figure 2 shows this character in its 36 major variations (3 widths, 6 weights, and 2 slants). Both of these were printed on an Anadex 9501 printer. Completion date for this alphabet is somewhat uncertain although a useful version should be finished this summer. While it is anticipated that Chel will be made available outside OCLC, no distribution mechanism has been established.

```
The letter b
File created by METAFONT 06/05/81 08:38:29 AM
"The letter b"
```

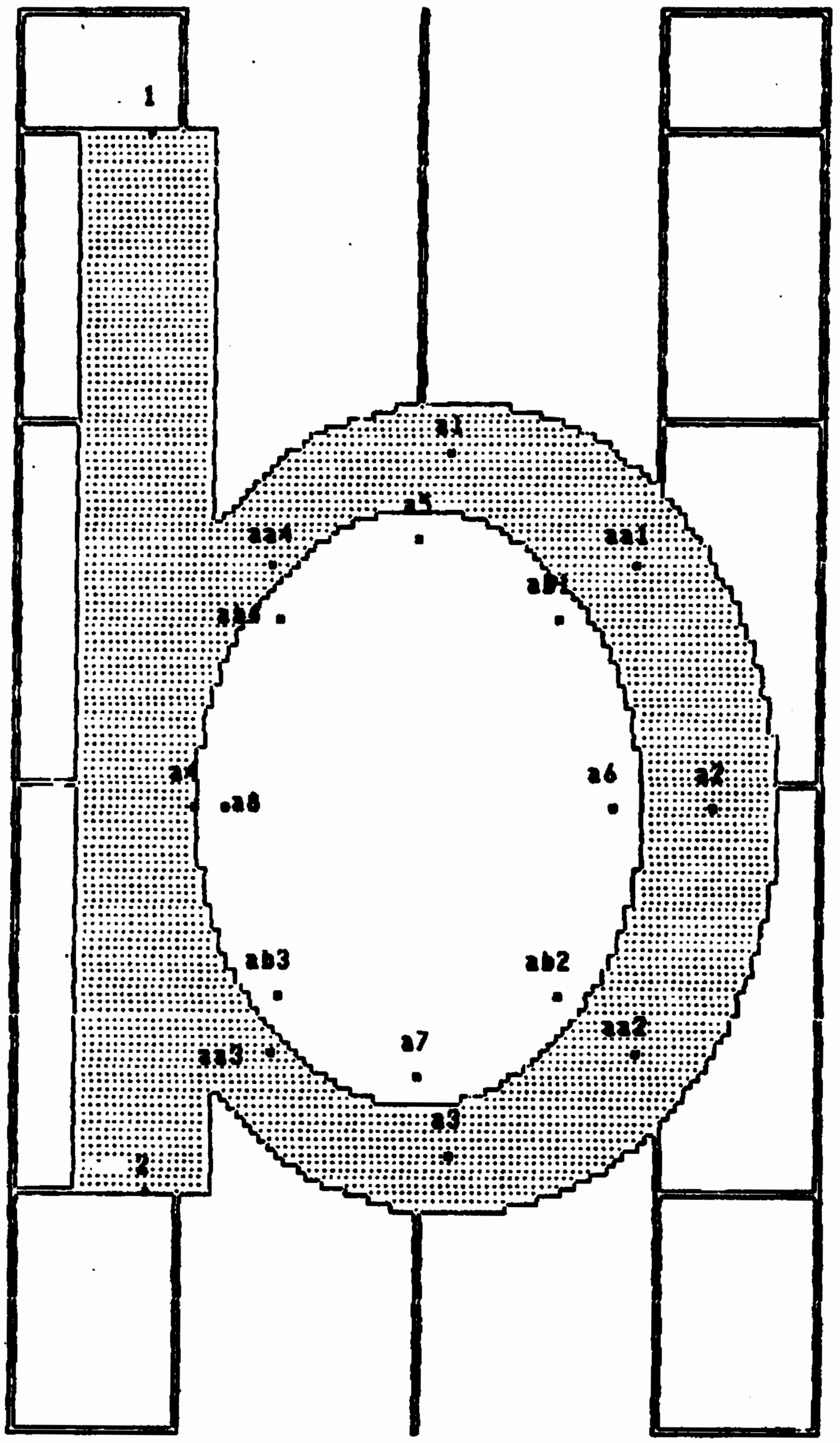


Figure 1. Lowercase "b": Proofmode at OCLC.

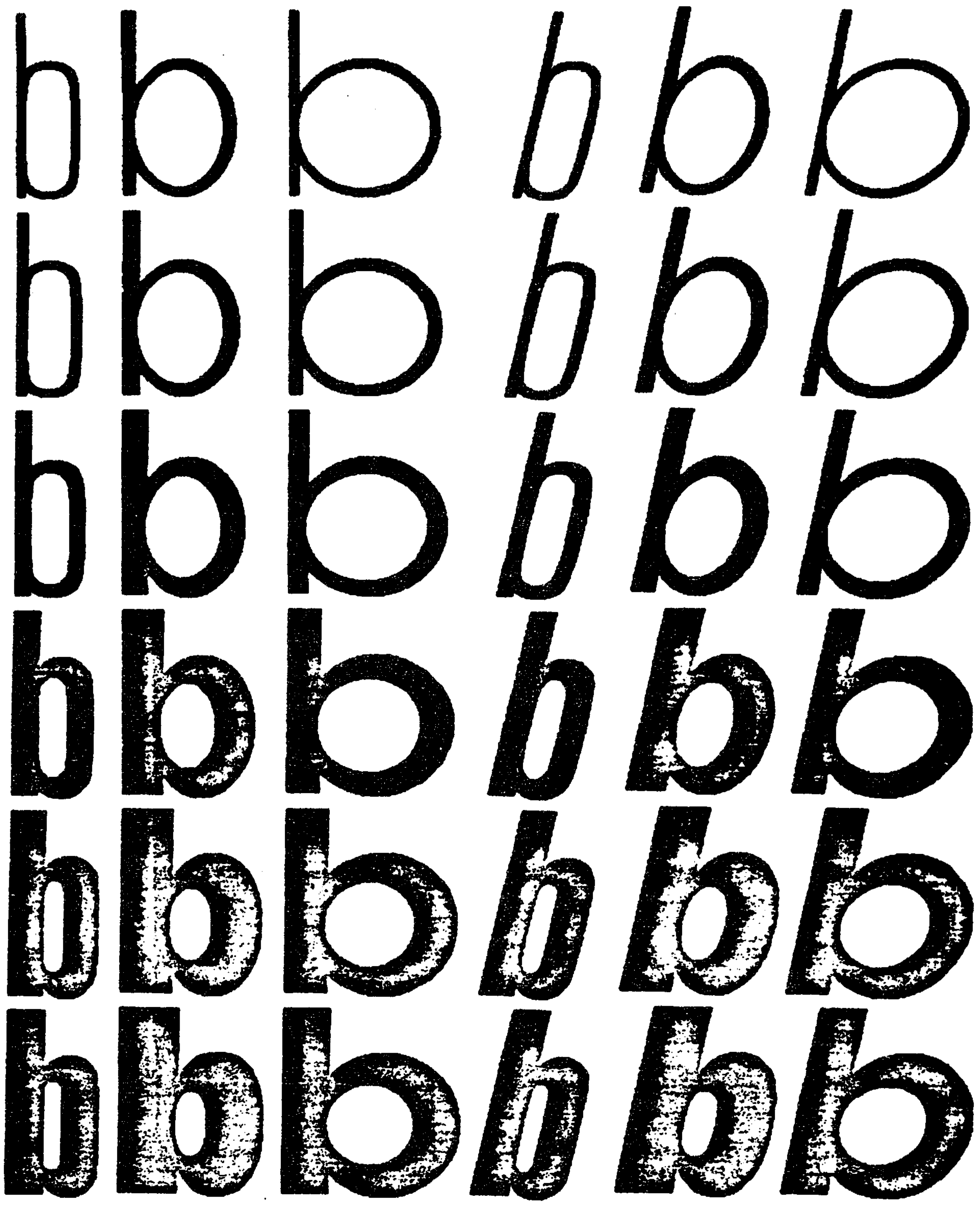


Figure 2. Variations on the letter "b": Chel font at OCLC.

## FONT DEVELOPMENT AT THE AMERICAN MATHEMATICAL SOCIETY

William J. LeVeque

As Don Knuth has declared, his primary purpose in developing **METAFONT** and TEX was to be able to produce new editions of his book that would look like the originals. So it was reasonable that he should focus his attention, while designing particular alphabets and character sets in the Computer Modern family, on those fonts needed for composing ordinary text and mathematics. Roughly speaking, these were Computer Modern Roman, Bold Face, Slant, Sans Serif, and math mode Italics, Greek, symbols and script, in 5, 6, 7, 8, 9, and 10 point sizes, and Upright Italic and Typewriter in 9 and 10 point sizes.

When the American Mathematical Society (AMS) decided to continue the development of Knuth's system so that it could be used for the production of a wide variety of Society publications, it was apparent that additional fonts and symbols would be needed. Here are examples of some of the gaps we saw:

- Mathematical Reviews uses over 250 distinct non-alphabetic mathematical symbols, of which fewer than 100 are available from Knuth's font tables.
- There is a large number of diacritical marks which are used in the various languages that Mathematical Reviews quotes references in, and they occur on a larger number of letters than one might expect; it is not feasible to reserve a place in a font table for each combination of letter with diacritic. (And of course mathematicians put accents on letters in entirely unpredictable ways.). If the accents are designed separately, then the letters should be vertical if they are to center properly under or over the accents. This requires, for example, an upright Greek alphabet.
- Mathematical Reviews uses the whole Cyrillic family, none of which had been designed.
- Various publications such as the Combined Membership List are produced in very small type (5 or 6 point), and these had been designed by Knuth only for sub- and superscripts, not for text. What was needed was a text face as easy to read as that in a telephone directory.

The AMS Trustees decided to create a Font Committee (actually a subcommittee of its Committee on Composition Technology) to work on this problem; this committee contains both AMS staff members and working mathematicians who have had prior experience or interest in type design. Also, they commissioned Hermann Zapf, one of the world's outstanding type designers, to design several fonts especially for the AMS which would be esthetically consistent with Computer Modern Roman. It was agreed that the augmented font family, containing not only Knuth's and Zapf's fonts but also a large collection of mathematical symbols already designed by an AMS staff member, Mrs. Phoebe Murdock, for other purposes, should be called the Euler family.

Dr. Zapf has now prepared drawings of four alphabets: upright Greek and Fraktur (German), upright script, and an entirely new kind of alphabet which will be referred to as "handwritten". The latter is rather like an upright italic, and will be used in place of Knuth's math mode italics. Zapf's drawings have been distributed to the Font Committee, and the resulting suggestions have led to modifications of several of the characters, to accord better with what mathematicians are accustomed to.

The next step is to get **METAFONT** programs written, to simulate these designs in 10-point type. This work, now being done by one of Knuth's graduate students, Scott Kim, should be completed within a few weeks.

One of Knuth's **METAFONT** programs automatically converts a "roman" version of a character (whatever that means, for Greek or Fraktur) to four others: slanted, sans serif, bold and typewriter. So these five variants will immediately be available for these characters and alphabets.

There is an additional step—or rather, there are many hundreds of additional steps—required to yield what could be regarded as a complete collection of fonts in the Euler family. As is well known to type designers, type faces do not satisfactorily scale up or down in size in a linear way. A 10-point font reduced photographically to 6 points does not look right; hair lines tend to vanish altogether, thick lines get too thick, upper case letters begin to dominate small lower case letters, etc. So it is necessary to adjust individually the various parameters that govern the relative sizes of these features, for each desired size. This is not a terribly time-consuming job, for a single alphabet—but when there are five alphabets, in five variant forms, in ten or fifteen sizes, plus all the mathematical symbols and their variations, the job looms large. This task is already under way in the Society office; it is expected to be about a year before all the font tables one could reasonably ask for are available, in the Euler family. But long before then, of course, the ones most urgently needed will have been completed, and it is hoped that TEX, and Michael Spivak's macro package, *AMS*-TEX, can be used for most of the composition work of the Society by the end of 1981.

At first these fonts will be for the Society's own use, but it is hoped that through licensing agreements the Euler family can be made generally available to other printing houses.

* * * * * * * * * * *

## TPHON

Ronald Whitney
American Mathematical Society

As was mentioned in Bill LeVeque's article (see page 39), the AMS has been using **METAFONT** to generate a set of small, highly condensed fonts to be used in the *Combined Membership List* of the Society and two other mathematical organizations. We started by creating a 6 point sans-serif Computer Modern font, and narrowed and thinned it until parameter twiddling no longer yielded significant improvements. Since proofmode was not yet available here and pens at Varian resolution were 1–2 pixels wide, more proofing than usual was done with the Alphatype. Inter-letter spacing seemed to provide the most difficulty.

As can be expected when a font family is pushed very far in a direction in which it is not designed to go, it became necessary for our purposes to bifurcate some parameters (serif correction, unit width, one pen size) and to modify some drawing routines. The result is our first approximation to what we have called the "telephone book" fonts. The TUG membership lists in this issue have been printed in TPHON and we welcome comments and criticisms about its readability.

* * * * * * * * * * *

## PROOFMODE AND MAGNIFICATION

Barry C. W. Doherty and Ronald F. Whitney
American Mathematical Society

A large part of the TEX effort at the AMS has been the development of a library of fonts adequate for our typesetting requirements, and for this we have been making extensive use of (the SAIL version of) METAFONT. So far this has taken four directions: (1) creation of fonts in larger point sizes (e.g., 14 and 18 point), (2) filling in holes (e.g., 8-point text italic), (3) production of a complete set of fonts for the Florida Data and (4) creation of a 'telephone book font' (cf. the article by Ron Whitney in this issue, p. 40).

All but (3) have necessitated changing the parameters in the .MF files, but only with the last have really significant changes had to be introduced, to compensate for the changes of parameters subjected to excessive shrinkage. While METAFONT has been able to generate the Florida Data fonts, it has not been able to cope with the rounding problems made more visible by the low resolution of the output; some improvement might have been had by modifying the .MF files—at the expense of inconsistency of fonts across devices—leaving as the only practical alternative the manipulation of the raster pattern via an intermediary file. Changing the font description would also violate Knuth's goal of designing to the highest resolution device, in our case the Alphatype CRS.

This development has been able to proceed without the availability of proofmode, although (4) concerns the creation of a new font, and would have been greatly facilitated had proofmode been available. Other pending work involves the creation of new fonts and new symbols. This work requires proofmode, or at least some means for examining an enlarged representation of the character.

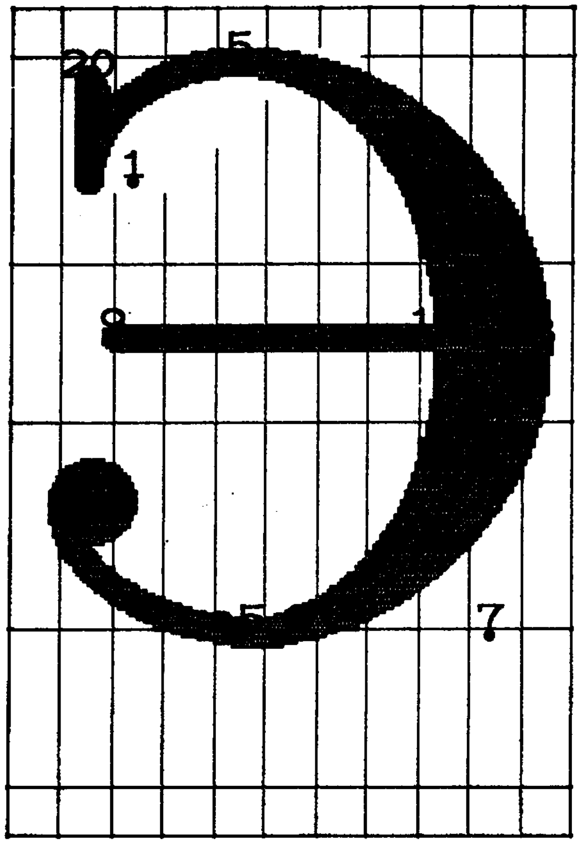
Proofmode was implemented originally for the Xerox XGP. The routines for proofmode reside in the METAFONT module MFOUT. Our task was to rework those routines to produce proofmode on the Varian, which meant the translation of XGP commands to those suitable for the Varian (or the Florida Data), requiring the translation of absolute movement commands to relative ones. That is, for the XGP one could say 'move to column 545' whereas for the Varian one has to say 'move 224 units from last reference point'. The job was made more difficult by the fact that the XGP is not a available commercially, and it was not immediately clear how to compare the XGP with the Varian.

The basic procedure was to make METAFONT proofmode write a TEX-like .DVI file, containing all appropriate commands and typeset characters. Full utilization of the DVI commands PUSH and POP to save and restore one's current position on the page prevented the need for the introduction of several real parameters to keep track of where one is on the page (and the amount by which this position had changed). This also prevented the occurrence of numerous (accumulative) rounding errors.

Changes were also necessary to the file-opening procedures to account for the fact that one was dealing with a binary rather than an ascii (text) file. At the same time the general file-handling procedures

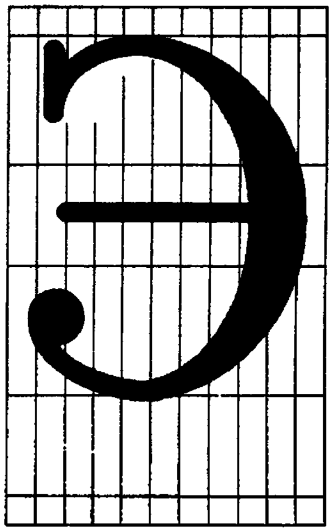
**July 1, 1981 09:38:10 CYRBUG.DVI.9 Page 1**

Cyrillic uppercase Э.



Varian proofmode at the AMS.

Varian output of magnified character.



**METAFONT** character description:

```
"Cyrillic letter Eprime";
call charbegin('037+3codeoffset,11,0,0,phh,0,rbowl);
cpen; top3y2=.3hh;
if bot3y2<.1hh: new y2; bot3y2=.1hh;
fi;
lft3x2=round .75u; w3 draw 2;                   % lower bulb
hpen; lft0x3=round 1.25u; y3=good6 .8hh; lft0x4=lft3x2; y4=y2;
x20=x3; rt4x3=rt0x1; y1=y3; top0y20=hh;
x5=.5[x3,x7]-u; x6=.5[x4,x7]; top0y5=hh+o; bot0y6=-o;
rt1x7=round(r-u); y7=y6;
if ucs$0: w0 ddraw 1..20, 3..20;                % upper serif
        rpen#; w4 draw 3{0,1}..5{1,0};          % erase spurious part
fi;
hpen; w0 draw 3{0,1}..5{1,0};                   % shoulder
call `a darc(5,7,w5);                           % bowl
y9=y10=.52hh; x9=2u; x10=lft5x7;
w0 draw 9..10;                                  % bar
w0 draw 6{-1,0}..4{0,1}.                        % tail
```

were changed to to allow greater flexibility in both specifying and reporting the full file names.

Changes were worked out first on an 'archaic' TEX version of METAFONT but have since been transferred to a more recent (TFM) version in a rather straightforward manner. Some modifications in approach have been suggested by work done at Stanford, although the development has been independent.

A METAFONT description is device-independent as well as point-size-independent—within limits: that limit is METAFONT itself with its highly device-dependent routines and its inability to deal (linearly) with the full range of point sizes of interest to the typesetting community. The METAFONT description may be device independent but METAFONT is not, and that sort of generality is not possible without the use of translators to modify METAFONT's output. The consequence of this last is crucial with respect to the creation of a font library, and its implications are discussed elsewhere in this issue.

The larger figure on the previous page is a Cyrillic Э produced by proofmode. Vertical unitlines and significant horizontal lines (representing h-height, x-height, axis-height, baseline, and descender depth, from top to bottom) form a grid on which the character is drawn. Points mentioned in the drawing routine are labelled.

In addition to proofmode (which takes work to implement), METAFONT contains a magnification option which is very easy to use. In ordinary font production for the Varian, one runs METAFONT and sets the variable `pixels` to 3.6. This value represents the number of pixels per point on that machine (increased by 30% since we use the Varian for proofing). To obtain the other character shown, one simply fools the Varian into thinking that one point (the printer's unit) contains 3.6*15 pixels. For the Stanford METAFONT this is done by specifying `mode=-1` (to tell METAFONT that you want a Varian font and that a magnification factor is coming) and `mag=15`. The same sort of procedure would work for any other output device for which METAFONT can produce characters.

Although significant points are neither indicated nor labelled with this magnification option, the figure obtained is of value in design. Subtleties of strokes are more likely to stand out here than in our Varian output of proofmode. It seems to us, for example, that the vertex at the bottom of the letter is more apparent in the smaller than in the larger figure.

* * * * * * * * * * * *

## Warnings & Limitations

* * * * * * * * * * * *

### Uppercase Update; Fickle Fonts

Last issue's warning about `\uppercase` has now been rendered obsolete by a change in TEX, whereby dimensions (`.5em`) and function words (`for`, `after`) are recognized in any combination of upper- and lower-case letters (`.5EM`, `For`, `aFtEr`). This change was made in Don Knuth's own version of TEX on February 27, 1981, and is described in the errata list among the "Extensions since last printing". But check the status of your local version before changing your macros!

Welcome though this change may be, it lays another trap: A macro for formatting entries in an index included the specification "... `\hangindent 10pt #1` ..." and an index entry "Forecasting" was rendered as "casting", indented just a bit more than the specified hanging indent. Don explains it thus:

"The word '`for`' is allowed after '`\hangindent`'; e.g.,

```
      \hangindent 3pt for 5
```

and moreover you can use letters as constants as in

```
      \char e     (= \char'145)
```

Then `\hangindent 3pt Forecasting` means

```
      \hangindent 3pt for '145 casting
```

Likewise, the word '`plus`' or '`minus`' will be gobbled after `\hskip 1pt` ..."

To disarm the trap, add a couple more braces to the offending macro: "... `\hangindent 10pt{}#1` ..."

. . . . . . . . .

Probably most sites use a version of TEX that has preloaded fonts, and probably most sites preload the fonts specified in `basic.tex` (see the TEX manual, Appendix B, for details). But some users, for whatever reason, decide to associate different fonts with the one-character names assumed by `basic`. When a file using non-`basic` font names is run through a version of TEX that has preloaded `basic`, the job may run to completion with no warning (the SAIL implementation of TEX gives no warning, but some Pascal versions may—at least the VAX/VMS version does so). TEX will use the metrics of the preloaded fonts, but the spooler will use the fonts requested by the user, and a ragged right margin may be only symptom.

There are two solutions: (1) Go back into the input files and change all conflicting font designators—this can get very messy; or (2) use a version of TEX in which no fonts have been preloaded; such a version, commonly known as "VIRGINTEX", will start up much more slowly than a preloaded version, owing to the greater number of font metric files that must be loaded at run time. The following convention has been adopted at many installations: Preloaded fonts use no capital letters. Thus you are always safe if you introduce a new font called A, B, ..., Z. (Actually, the AMS requires an extended set of fonts, including a full complement of cyrillic fonts in 6 sizes; these are called A, ..., F, but G through Z remain open for special use.)

Barbara Beeton

* * * * * * * * * * *

## MACRO COLUMN

*Send Submissions to:*
*Lynne A. Price*
*TUG Macro Coordinator*
*Calma R&D*
*212 Gibraltar Dr.*
*Sunnyvale, CA 94086*

The macro column is a new regular feature of TUGboat. It is a forum where TEX users can exchange formatting problems (with or without solutions), questions about writing macros, comments on macros published in earlier issues of TUGboat, etc.

* * *

Discussion of macros at the TEX Implementors' Workshop in May included some simple suggestions for increasing portability of macros across TEX sites. First, the excellent suggestion was made that ASCII sites attempt to standardize the characters chosen to replace SAIL delimiters. The AMS-TEX conventions are recommended: ampersand (&) for the tab character, underscore (_) for the subscript indicator, and caret (^) for the superscript delimiter. Second, macro packages typically include several font declarations. Incompatible assignment of font codes makes it difficult for users to select an assortment of macros from different packages. If font codes assigned in a macro file do not correspond to the fonts preloaded by some versions of TEX, strange results can be difficult to explain. There is no total solution to this problem, but it can be minimized. Macro packages should come with documentation describing the fonts and font codes used. When sending files to another installation, users should remember that preloaded fonts differ from site to site. A helpful convention in assigning font codes is to reserve uppercase letters for user declarations and to let standard macro packages use other characters. Patrick Milligan's DefineFont macro described below can be used to automatically assign available font codes.

* * * * * * * * * * *

### Macros on Microfiche

*Editor's note: In an effort to hold down expenses, some of the more extensive macro packages in future issues of TUGboat will be published on microfiche, with a summary or introduction to each package included in this column. Authors of macro packages who submit their work for publication here are requested to supply such an introduction along with the camera copy of the package. Because fiche is not as easy to use as paper, an attempt will be made to arrange for the collection and distribution of these macro packages in machine-readable form (probably on magnetic tape); details will be published as soon as they are known. Fiche will conform to the following specifications: negative image (white characters on black), 105mm × 148mm, 24-to-1 reduction ratio, containing 98 frames per fiche.*

* * * * * * * * * * *

### ERRATUM: NOFILL PROGRAM

Patrick Milligan
BNR Inc.

There was one subtle error in the program listing of both the SAIL and Pascal versions of the NOFILL program that appeared in TUGboat Vol 2, No. 1. In both programs, the definitions of macros \´ and \` were reversed (see pages 90 and 96). As printed, the definitions are correct, but the program source was incorrect. Since the program source was run through NOFILL for publication, the incorrect definitions became correct, but all other uses of ´ (acute accent) and ` (grave accent) were incorrect.

Also, there was some confusion about the table of contents entry on page 136 entitled *NOFILL Program with Pascal Source.* When the two programs were submitted to TUGboat, it was not clear if the SAIL version would be printed, or the Pascal version, or both. The introduction to the SAIL version was appropriate to both versions, but no introduction was prepared for the Pascal code.

**READFONTINFO**

This is an integer function that has the following parameters:

| Var.? | *Name* | *Type* |
|---|---|---|
| | FYL | INTEGER |
| Var | FONTINFO | FNTINFOARRAY |
| Var | FMEM | FMEMARRAY |
| Var | WDBASE | FBASEARRAY |
| Var | HTBASE | FBASEARRAY |
| Var | DPBASE | FBASEARRAY |
| Var | ICBASE | FBASEARRAY |
| Var | LGBASE | FBASEARRAY |
| Var | KRBASE | FBASEARRAY |
| Var | EXTBASE | FBASEARRAY |
| Var | PARBASE | FBASEARRAY |
| Var | FCKSUM | FBASEARRAY |
| Var | FPFB | FBASEARRAY |
| Var | FSIZE | FSIZEARRAY |
| Var | FPFI | FPIARRAY |
| Var | FMEMPTR | INTEGER |
| Var | PSIZE | REAL |
| Var | ATCLAUSE | BOOLEAN |

Reads font information from file FONTFIL. The integr FYL is used as an index in the various array parameters to establish the destination of this information.

**RELEASE**

Procedure with one parameter.

| *Name* | *Type* |
|---|---|
| FYL | INTEGER |

The integer FYL must be in the range [1..6]. It selects one of ICHAN1 through ICHAN6 and executes RESET(ICHANx) followed by FILPTR:=FILPTR-1.

This closes and releases the indicated file and frees the entry in FILENAME.

**RSETFILE**

Procedure with four parameters.

| *Name* | *Type* |
|---|---|
| ID | INTEGER |
| FNAME | CHAR9 |
| FDIRECTORY | INTEGER |
| FDEVICE | CHAR6 |

The integer *ID* must be in the range [1..6]. It selects one of ICHAN1 through ICHAN6 to be opened for input and associates it with FNAME, FDIRECTORY, and FDEVICE.

```
% Examples:
%       \DefineFont{cmtt}{\tt}          % typewriter font
%       \DefineFont{mflogo}{\mflogo}    % METAFONT logo font

\def\DefineFont#1#2{
  \if !\UserFonts!{
    \xdef#2{}\send9{Error: No font codes available for font #1}}
  \else{
    \Apply {\First} to {\UserFonts!} -> {\FontCode} % Get font code
    \font\FontCode=#1                              % Load font
    \xdef#2{\curfont \FontCode}                    % Define macro to use font
    \Apply {\Rest} to {\UserFonts!} -> {\UserFonts} % Remove code from list
  }
}

% The \Apply macro is used to apply a macro to its argument, when the
% argument is a macro also.  The trick is to fool \TEX into expanding
% the argument before the macro is applied.  If a better way exists to
% perform this feat, please send your solution to TUGBoat.
%
% Usage:
%       \Apply {<function>} to {<argument>} -> {<result>}
%
% where:
%       <function>      is the macro to apply
%       <argument>      is the macro containing the argument to <function>
%       <result>        is the macro used to save the result
%

\def\Apply #1 to #2 -> #3{
  \let\Func=\let                % Setup dummy function
  \xdef\Eval{\xdef#3{\Func #2}} % Expand argument
  \let\Func=#1                  % Redefine function to use macro
  \Eval                         % Apply macro to its argument
}

% The \First and \Rest macros are used to manipulate strings terminated with
% an exclamation mark (!).

\def\First#1#2!{#1}     % Returns first character of string
\def\Rest#1#2!{#2}      % Returns string with first character removed

% The macro \UserFonts describes the set of font codes available to
% \DefineFont.  The list of font codes should not contain an exclamation
% mark (!) since this is used to terminate strings passed to the \First
% and \Rest macros (and it isn't a valid font code anyway).  A reasonable
% convention for font codes is to have all upper case letters available
% for user fonts:

\def\UserFonts{ABCDEFGHIJKLMNOPQRSTUVWXYZ}

% If \DefineFont is used to allocate all fonts used (including those in
% BASIC), then all 64 possible font codes should declared.
```

# A MACRO MENAGERIE

Brendan D. McKay

## 1. Math-style testing

One of the things which one should be able to do in TEX, but which apparently is impossible, is to test for the current math-style (display, text, script or scriptscript). For example, how does one write a macro which produces a bold-face "g" of the right size? (`\def\g{\hbox{\bf g}}` obviously doesn't.) Here's a trick which doesn't completely solve the problem, but which goes a long way towards that goal.

```
\xdef\subscr{↓}  \xdef\supscr{↑}
\chcode'1=13 \chcode'136=13
\def\MS{T}
\def↓#1{\subscr{\if\MS T{\def\MS{S}#1}\else
   {\def\MS{X}#1}}}
\def↑#1{\supscr{\if\MS T{\def\MS{S}#1}\else
   {\def\MS{X}#1}}}
```

The idea is to maintain a macro `\MS` which has the value "S" for scriptstyle; "X" for scriptscript-style, and "T" for all other styles, including non-math mode. This can then be tested using the `\if` macro. The definitions above will maintain `\MS` correctly if the style changes by use of the subscript or superscript characters, but not otherwise. Style change macros like `\scriptstyle` can also be redefined to maintain `\MS`, but automatic changes caused by things like `\over` will go undetected. In such cases the user must define `\MS` himself, if it is going to be tested.

As a sample application, here is a definition of `\bf` ("select bold-face") which behaves the same as normal in non-math mode and selects a font of the right size in math-mode. In the latter case it acts only on the following character, control sequence or group. Let's suppose that A, B and C are bold-face fonts of the required sizes.

```
\def\bf{\ifmmode{\gdef\Fnt##1{\hbox
   {\if\Ms T{\:A##1}\else
   {\if\MS S{\:B##1}\else{\:C##1}}}}}\else
   {\gdef\Fnt{\:A}}\Fnt}
```

In our second application we'll define a macro for raising a portion of text. If you type "`\lift⟨text⟩\by⟨dimen⟩\\`", then ⟨text⟩ is put in a `\hbox` and raised by an amount ⟨dimen⟩. ⟨text⟩ will appear in the current style, except that display style and text style are not distinguished.

```
\def\lift#1\by#2\\{\raise#2\hbox{\ifmmode
{\if\MS T{$#1 $}\else
   {\if\MS S{$\scriptstyle#1$}\else
   {$\scriptscriptstyle#1$}}}\else{#1}}}
```

## 2. Groupless \ifs

A good source of inscrutable bugs involves the way that TEX handles conditionals like `\if`, `\ifpos`, `\ifvmode` etc.. Let's suppose that we want to select font A if the macro `\format` has the value "1" and font B otherwise. The obvious method is

```
\if\format1{\:A}\else{\:B}
```

but this doesn't work. The reason is that the text produced is not "`\:A`" or "`\:B`", but "`{\:A}`" or "`{\:B}`". Since font definitions are revoked at the end of groups the total effect is ⟨nothing useful⟩. It is sometimes handy to have another version of `\if` which avoids this rather unsatisfactory state of affairs. While we're at it, we'll change the format to

```
\If⟨char₁⟩⟨char₂⟩⟨true text⟩\else
   ⟨false text⟩\endif
```

— a few fewer braces never hurt anybody. Three possible definitions for `\If` are as follows.

```
(1) \def\If#1#2#3\else#4\endif
    {\if#1#2{\gdef\Iftemp{#3}}\else
    {\gdef\Iftemp{#4}}\Iftemp}
(2) \def\else#1\endif{}  \def\endif{}
    \def\If#1#2{\if#1#2{\gdef\Iftemp{}}\else
    {\gdef\Iftemp##1\else{}}\Iftemp}
(3) \def\If#1#2{\if#1#2{\gdef\Iftemp##1\else
    ##2\endif{##1}}\else{\gdef\Iftemp##1\else
    ##2\endif{##2}}\Iftemp}
```

All three definitions work in most ordinary circumstances. The first definition has the unpleasant peculiarity that any #s which occur in ⟨true text⟩ or ⟨false text⟩ must be typed as ##, a problem which grows exponentially if `\If`s are nested. The second definition avoids this problem but has another deficiency: it won't nest properly (*why?*). The third definition avoids both problems. If ⟨true text⟩ or ⟨false text⟩ contains another `\If`, simply enclose it in {}s. This doesn't cause grouping, of course, but it will ensure that each `\else` or `\endif` gets paired with the right `\If`.

## 3. Recursion

Although the TEX manual apparently never says so, the macro facility in TEX is completely recursive. In other words, macros can directly or indirectly call themselves. Of course, we are not given this little gem of information because the knowledge would be almost useless. Nevertheless, there is a little gap between "almost useless" and "completely useless", and this section is devoted to exploring it. Three applications of recursion we will consider are (i) loop structures, (ii) counter arithmetic and (iii) macros accepting variable numbers of arguments.

(i) Loops are quite easy to create in TEX as long as one respects TEX's finite capacity. In order to make loops which can be repeated a large number of times, the recursive call must be the very last thing in the expansion, and in particular it must not be in a group (TEX won't nest groups to an indefinite depth). The last requirement means that the recursive call can't be part of the result text of a conditional (see Section 2). Here's some examples:

```
\def\savecount#1#2{\ifpos#1{\xdef
   #2{\count#1}}\else
   {\setcount#1-\count#1\xdef#2{-\count#1}
   \setcount#1-\count#1}}
\gdef\Wtemp#1#2{#2\Wloop#1{#2}}
\def\Wloop#1#2{\ifpos#1{}\else
   {\gdef\Wtemp##1##2{}}\Wtemp#1{#2}}
\def\while#1#2\endwhile{\Wloop#1{#2}\gdef
   \Wtemp##1##2{##2\Wloop##1{##2}}}
\def\repeat#1\times#2\endrepeat
   {\savecount9\Rtemp\setcount9#1
   \while9{#2\advcount9by-1}\endwhile
   {\setcount9\Rtemp}}
```

\savecount⟨digit⟩⟨control sequence⟩ saves the value of a counter in a control sequence. \while⟨digit⟩⟨text⟩\endwhile will produce ⟨text⟩ repeatedly until \count⟨digit⟩ becomes non-positive. (Presumably ⟨text⟩ will set the counter non-positive eventually.) \whiles can be nested if they use different counters.

\repeat⟨value⟩\times⟨text⟩\endrepeat will produce ⟨text⟩ precisely ⟨value⟩ times, where ⟨value⟩ can be either a number or a counter. The use of \Rtemp in \repeat enables \repeats to be nested to one level, but no further. For example, \repeat5\times{x-\repeat3\times aA\endrepeat}\endrepeat produces

x-aAaAaAx-aAaAaAx-aAaAaAx-aAaAaAx-aAaAaA.

(ii) The fact that counter operations like multipli.ation and division are not provided by TEX is one indication of their likely usefulness. Of course, that won't stop us from doing these operations anyhow.

```
\def\neg#1{\setcount#1-\count#1}
\def\Hlf#1#2 {\advcount9by-#2\advcount9by-#2
   \ifpos9{\advcount#1by#2}\else
   {\advcount9by#2\advcount9by#2}}
\def\halve#1{\savecount9\Htemp
   \setcount9\count#1\advcount9\setcount#1 0
   \Hlf#11073741824 \Hlf#1536870912
   \Hlf#1268435456 \Hlf#1134217728
   \Hlf#167108864 \Hlf#133554432 \Hlf#116777216
   \Hlf#18388608 \Hlf#14194304 \Hlf#12097152
   \Hlf#11048576 \Hlf#1524288 \Hlf#1262144
   \Hlf#1131072 \Hlf#165536 \Hlf#132768
   \Hlf#116384 \Hlf#18192 \Hlf#14096
   \Hlf#12048 \Hlf#11024 \Hlf#1512 \Hlf#1256
   \Hlf#1128 \Hlf#164 \Hlf#132 \Hlf#116 \Hlf#18
   \Hlf#14 \Hlf#12 \Hlf#11 {\setcount9\Htemp}}
\def\multiply#1\into#2{\setcount8#1\setcount9
   \count#2\setcount#2 0
   \while8\ifeven8{}\else
   {\advcount#2by\count9}
   \advcount9by\count9\halve8\endwhile}
\def\divide#1\into#2{\setcount9\count#2
   \setcount#2-1\advcount9
   \while9{\advcount#2by1
   \advcount9by-#1}\endwhile}
\def\Divide#1\into#2{\ifpos#2{\divide#1\into
   #2}\else{\neg#2\divide#1\into#2\neg#2}}
\def\sqroot#1{\setcount9\count#1\advcount9
   \setcount#1-1\setcount81
   \while9{\advcount9by-\count8
   \advcount#1by1\advcount8by2}\endwhile}
```

\halve⟨digit⟩ will divide any counter other than counter 9 by two, provided its original value is in the range 0 to 4294967294. Some of the earliest calls to \Hlf will need to be removed for machines with small word-sizes. \sqroot⟨digit⟩ will take the square root of any non-negative counter other than counter 8 or 9. In the other cases, the format is \operation⟨value⟩\into⟨digit⟩, where ⟨value⟩ is a number or a counter and ⟨digit⟩ is a counter number for the other argument and the answer. ⟨value⟩ must be non-negative in each case. \count⟨digit⟩ may be negative for \Divide or \multiply but not for \divide. The restrictions on which counters can't be used and which counters are destroyed are most easily seen by examining the definitions. Both \halve and \multiply are quite fast, but \divide, \Divide and \sqroot take time proportional to the answer.

(iii) The method by which recursion can allow a macro to apparently accept any number of arguments is best illustrated by an example. The macro \options below will accept any number of single character arguments, each of which will presumably cause some useful action. If an "x" occurs it must be followed by two arguments (which somehow belong to the x). Also, a "d" implies a "j" as well. The end of the argument list is indicated by a period. A possible call would be "\options rx{30pt}{75pt}d.".

```
\def\options#1{def\Next{\options}
   \If a#1⟨something⟩\else\endif
   \If j#1⟨something⟩\else\endif
   \If x#1
    \def\Next##1##2{⟨something⟩\options}
    \else\endif
   \If d#1⟨something⟩
    \options j. \def\Next{\options}\else\endif
   \If .#1\def\Next{}\else\endif
   \Next}
```

A macro of this sort is invaluable in writing a general purpose macro package, especially one to be used by many people. A large number of different style options can be provided, and each user can easily select any combination.

*Research Problems:*

(1) Speed up \divide and \sqroot.

(2) Write a macro which tests two character strings for a character in common. Then dream up an application.

## 4. Pictures

In this section we describe a few macros which can facilitate the drawing of complicated diagrams. The two macros at the heart of the method are these:

```
\def\picture#1#2#3#4\endpicture{{\varunit#1
   \vbox to #2{\vss\hbox to #3{\!#4\hss}}}}
\def\put#1(#2,#3){\raise#3vu\hbox to 0pt
   {\hskip#2vu#1\hss}\!}
```

The result of `\picture`⟨`dimen`$_1$⟩⟨`dimen`$_2$⟩ ⟨`dimen`$_3$⟩⟨`hlist`⟩`\endpicture` is a vertical box of height ⟨`dimen`$_2$⟩ containing a horizontal box of width ⟨`dimen`$_3$⟩ which contains ⟨`hlist`⟩. ⟨`dimen`$_1$⟩ becomes `1vu`. Each position in the picture has coordinates of the form $(x, y)$, where $x$ is the distance in vu from the left boundary and $y$ is the distance in *vu* from the bottom of the picture. Thus $(0, 0)$ is the reference point of the picture. To put ⟨`something`⟩ at position (36, 475) simply type `\put`⟨`something`⟩`(36,475)`. The second coordinate cannot be negative. Both coordinates can be specified as the values of counters.

By putting `\put`s inside `\put`s, a temporary change of origin can be affected, allowing sections of the picture to be moved around in one piece. For even greater flexibility, picture a `\picture` within a `\picture`. (The inside `\picture` should be given width zero.) The overall scale of the picture can be adjusted by changing ⟨`dimen`$_1$⟩.

Just for fun, we'll give macros for inserting horizontal or vertical rules into a picture and for drawing dotted lines.

```
\def\line(#1,#2)(#3,#4){\put\setcount8#4
   \advcount8by-#2
   \ifpos8{\hskip-0.2pt
   \vrule depth0pt width0.4pt height \count8vu}
   \else{\setcount8#3\advcount8by-#1
   \vrule depth0.2pt width \count8vu
   height 0.2pt}(#1,#2)}
\def\speck{\hskip-0.3pt
   \vrule height0.3pt depth0.3pt width0.6pt}
\def\dotline#1(#2,#3)(#4,#5){\put
   \speck(#2,#3)\setcount7#4
   \advcount7by-#2\setcount8#5\advcount8by-#3
   \Divide#1\into7\Divide#1\into8
   \setcount5#2\setcount6#3
   \repeat#1\times\advcount5by\count7
   \advcount6by\count8
   \put\speck(\count5,\count6)\endrepeat\!}
```

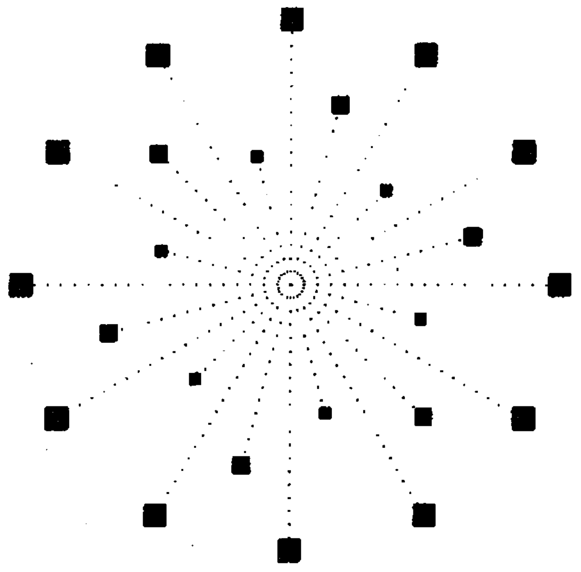
`\line(`⟨`coords`$_1$⟩`)(`⟨`coords`$_2$⟩`)` will draw a solid line between the points given. These must be specified in the order left–right for a horizontal rule and bottom–top for a vertical rule.

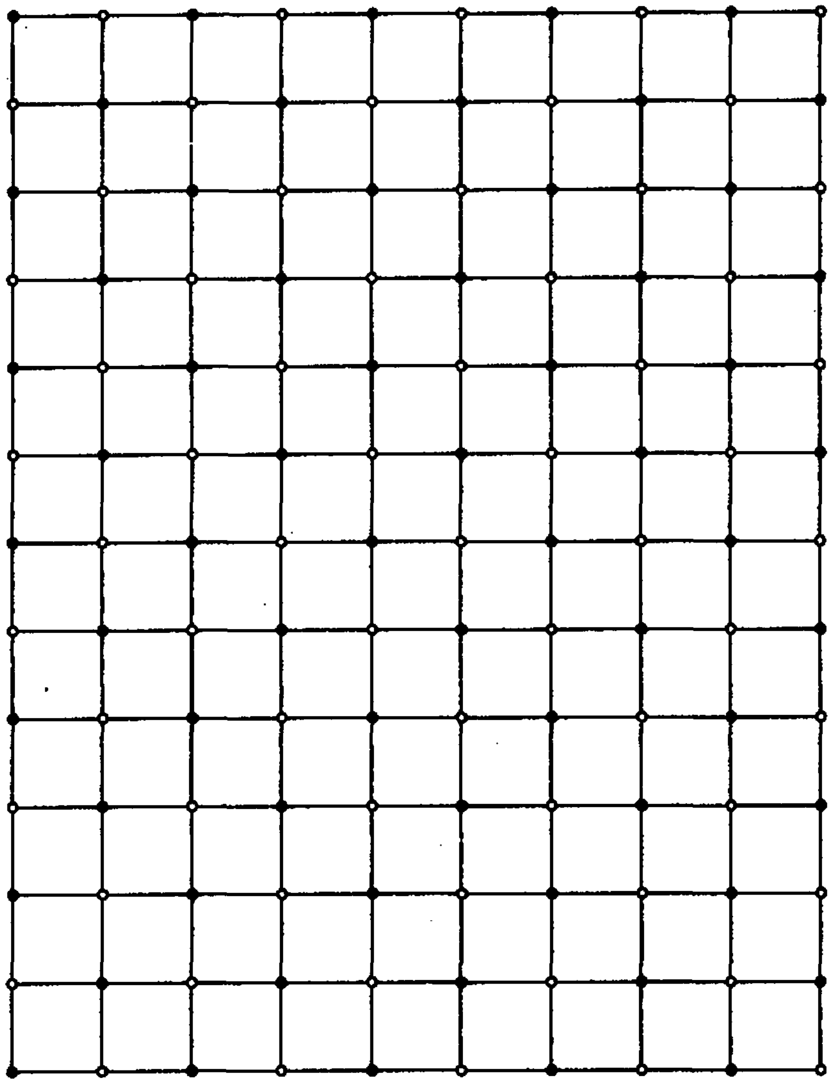
`\dotline`⟨`value`⟩`(`⟨`coords`$_1$⟩`)(`⟨`coords`$_2$⟩`)` will draw a dotted line consisting of ⟨`value`⟩+1 `\speck`s between the points specified, which can be given in either order. The last `\speck` can be misplaced by up to ⟨`value`⟩vu due to rounding error, so 1vu should be small if ⟨`value`⟩ is large. `\dotline` can be used to make solid diagonal lines by placing many small dots very close together, but you won't get far before TeX runs out of space. Both `\line` and `\dotline` will only accept integer coordinates, but this is no restriction if 1vu is small.

`\picture` can also be used as a very versatile and simple to use system for creating complicated symbols, like .

We conclude with a couple of more complicated `\picture`s. Here is the source for the second:

```
\def\overt{\lower2.5pt\hbox
   {\hskip-2.3pt\:u\char'5}}
\def\cvert{\lower2.5pt\hbox
   {\hskip-2.3pt\:u\char'17}}
\picture{0.083pt}{size}{size}
   \setcount4 5000\setcount5 4980
   \setcount6 4620\setcount7 0
   \while4{\setcount3 3800\setcount2 3780
   \setcount1 3420
   \while3\ifeven7
   {\put\overt(\count3,\count4)}\else
   {\put\cvert(\count3,\count4)}\ifpos1{\line
   (\count1,\count4)(\count2,\count4)}\else{}
   \ifpos6{\line
   (\count3,\count6)(\count3,\count5)}\else{}
   \advcount7\advcount1by-400\advcount2by-400
   \advcount3by-400\endwhile
   \advcount7\advcount4by-400\advcount5by-400
   \advcount6by-400}\endwhile
\endpicture
```





* * * * * * * * * * * * *

## MACRO MADNESS

Michael Spivak
2478 Woodridge Drive,
Decatur, GA 30033

This article is an extract from the documentation for the not-yet-completed AMS-TeX macro package. It discusses certain tricks and pitfalls that other macro writers might want to know about. Needless to say, none of this trickery would have been possible without the help of Don Knuth.

It should be mentioned that the AMS-TeX macro package initially \chcodes the symbol | (ASCII ´174) to be a letter, and all internal AMS-TeX macros contain a | as one of their letters. At the very end of the macro file, | is re-\chcoded to be of type 12, so that the AMS-TeX user cannot redefine, or even use, these control sequences (the input \cs| will be read as \cs |). For convenience, we will omit the |s here, and we will use mnemonic names for control sequences—the actual names used by AMS-TeX are very short (at most three letters, including any |s), in order to preserve memory space.

Please report any bugs to the above address as soon as possible—before the macro package gets distributed widely!

### I. Branching Mechanisms.

The only branching mechanism provided by TeX is

```
\if ⟨char_1⟩⟨char_2⟩{⟨true text⟩}
\else{⟨false text⟩}
```

and its relatives. Unfortunately, there are certain peculiarities of \if...\else that require special care.

(a) An \if...\else construction is processed in TeX's "digestive system", rather than in its "mouth". Suppose, for example, that we have two control sequences \csa#1 and \csb#1#2, taking one and two arguments, respectively, and a control sequence \flag that is sometimes defined to be T and sometimes defined to be F. We would like to define \cs to be \csa if \flag is T, and \csb if \flag is F [the argument(s) for \cs will simply be whatever comes next in the input text]. If we try to define

```
\def\cs{\if T\flag{\csa}\else{\csb}}
```

then a use of \cs will produce the error message

```
! Argument of \csa has an extra }
```

because TeX sees the } as soon as it looks for the argument after \csa or \csb. The solution to this problem is to define

```
\def\cs{\if T\flag{\gdef\result{\csa}}
        \else{\gdef\result{\csb}}
        \result}
```

◇ A similar problem arises in the following situation. Suppose that we have two different macro files, mfile.1 and mfile.2, and the value of \flag is supposed to determine which file to use (such a scheme is useful for saving TeX memory space). A definition like

```
\def\cs{\if T\flag{\input mfile.1}
      \else{\input mfile.2}...}
```

gives a different error message:

```
! Input page ended on nesting level 1
```

but the basic problem (and the solution) is exactly the same.

◇◇ If we make the definition

```
\def\If#1\then#2\else#3{\if#1
      {\gdef\result{#2}}
      \else{\gdef\result{#3}}\result}
```

then we can safely use constructions like

```
\If T\flag\then...\else{...}
```

The token \then is made part of the syntax of \If so that we can have constructions like \If \cs a\cs b\then..., where \cs#1 is a control sequence with one argument.

(b) Although ⟨char$_1$⟩ and ⟨char$_2$⟩ may be specified by control sequences like \flag, which TeX expands out, they cannot involve \if...\else again. Suppose, for example, that we have already defined

```
\def\ab#1{\if#1a{T}
          \else{\if#1b{T}\else{F}}}
```

so that \ab#1 is T if #1 is a or b, and F otherwise. We would now like to define \cs#1 to be ⟨true text⟩ if #1 is a or b, and ⟨false text⟩ otherwise. We cannot conveniently define

```
\def\cs#1{\if T\ab#1{⟨true text⟩}
          \else{⟨false text⟩}}
```

If we do this, then the input \cs x will become

```
\if T\if xa{T}\else{\if xb{T}
  \else{F}}{⟨true text⟩}\else{⟨false text⟩}
```

which causes TeX to try to compare T with \if, giving an error message.

Of course, the test for #1 being a or b could be made part of the definition of \cs, but the following scheme is far more advantageous:

```
\def\ab#1{\if#1a{\gdef\Ab{T}}
          \else{\if#1b{\gdef\Ab{T}}
               \else{\gdef\Ab{F}}}}
\def\cs#1{\ab#1
          \if T\Ab{⟨true text⟩}
          \else{⟨false text⟩}}
```

(c) In an \if...\else construction, ⟨char$_1$⟩ and ⟨char$_2$⟩ are supposed to be single characters (of type

0 to 12), or defined control sequences, possibly with arguments, that expand out to a character. So we can't use an \if...\else construction in a situation where we don't know for sure what the next input text will be. Suppose, for example, that \cs#1 is supposed to be ⟨true text⟩ if #1 is a comma, but ⟨false text⟩ otherwise. If we define

```
\def\cs#1{\if#1,{⟨true text⟩}
                    \else{⟨false text⟩}}
```

there is always the possibility that our input text will contain

```
\cs ...
```

where ... is a token that can't be used with \if, or even worse, a group {...}, which might produce total chaos. In order to deal with this we will use several tricks, which are also useful in other situations.

## II. Basic Kludges

Consider the definitions

```
\def\false#1{\gdef\ans{F}}
\def\tricka{A}
\def\trickb{B}
\def\trickc#1{\if#1A
        {\gdef\result{\false}}
        \else{\gdef\result{\gdef\ans{T}}}
        \result}
\def\empty#1#2\tricka{\trickc}
```

The control sequence \trickc will be used only in situations where the \if is safe. In fact, \trickc will arise only from an occurrence of \empty, and the control sequences \tricka, \trickb, \trickc and \empty will be used only in the construction

```
\empty...\tricka\tricka\trickb
```

Here ... will be some input text, with perhaps a few special $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX control sequences thrown in, but ... will never involve \tricka (remember that \tricka is really \tricka|, so it can't appear in a user's file).

We have to consider two possibilities for ... in order to determine the result of this construction. Suppose first that ... is not empty. Then argument #1 for \empty will be the first token or group of ... and argument #2 will be whatever remains (if anything). Hence

```
\empty...\tricka\tricka\trickb →
        → \trickc\tricka\trickb →
        → \false\trickb → \gdef\ans{F}
```

But suppose that ... is empty, so that we have

```
\empty\tricka\tricka\trickb
```

Note that argument #1 for \empty must be *non-empty*, since it is not followed by a token in the definition of \empty. So in the present case argument #1 for \empty will be the first \tricka. Consequently, the *second* \tricka will play the role of the token \tricka in the definition of \empty (and argument #2 will be empty). Thus

```
\empty\tricka\tricka\trickb →
      → \trickc\trickb → \gdef\ans{T}
```

In other words,

> \empty...\tricka\tricka\trickb
> defines \ans to be T if ... is empty,
> and F otherwise.

There would appear to be one exception to this rule: If ... is a blank space, or a sequence of blank spaces, then \ans will still be defined to be T, since spaces after the control sequence \empty are ignored. But in practice ... will always be an argument from some other macro, and in this case the exception does not arise. Suppose, for example, that we define

```
\def\try#1{\empty#1\tricka\tricka\trickb}
```

so that \try{#1} will test whether #1 is empty or not. If we give TEX the input

```
\try{ }
```

then the braces will be removed from { }, so this will be translated into

```
\empty␣\tricka\tricka\trickb
```

But in this situation the space indicated by ␣ is *not* ignored, so \ans will be defined to be F.

We might have arranged for the result of the combination \empty...\tricka\tricka\trickb simply to be T or F, rather than defining \ans to be T or F. But if we did this, a construction like

```
\if T\empty#1\tricka\tricka\trickb{...}
\else{...}
```

wouldn't work, because TEX would think that we were trying to compare T with the result of \empty#1\tricka.

The following variant of \empty is also useful:

```
\def\emptygp#1\endd
        {\empty#1\tricka\tricka\trickb}
```

Then

> \emptygp...\endd
> defines \ans to be T if ... is empty or {},
> and F otherwise.

It will be convenient to use the same flag \ans for the result of several of our macros. This won't produce problems if we ever have to perform two tests on two different arguments: we can always first use \empty, then \let\firstans=\ans, then use \emptygp, etc.

We also want to be able to check if `...` is a single token or group, rather than a string of several tokens or groups. One idea is to consider `\single...\endd` where `\single#1#2\endd` checks whether `#2` is empty:

```
\def\single#1#2\endd
          {\empty#2\tricka\tricka\trickb}
```

This won't quite work, since `...` might be something like `⟨token⟩{}`; in this case `#2` appearing in `\empty#2\tricka\tricka\trickb` will still be empty, since TeX removes an outer set of braces from any argument. So to be on the safe side, we add some extraneous character after `...` and let `\single#1#2#3\endd` check if `#3` is empty:

```
\def\single#1#2#3\endd
          {\empty#3\tricka\tricka\trickb}
```

Then

> `\single...*\endd`
> defines `\ans` to be `T` if `...` is a single token or group, and `F` otherwise.

Before using `\single...*\endd` it is essential to check that `...` isn't empty. Otherwise there will be problems, because of the very considerations that made `\empty` work. (An `\empty` check could be incorporated into the definition of `\single`, but whenever AMS-TeX uses `\single` a separate check has to be made anyway.)

As in the case of `\empty`, a space may legitimately occur as argument `#1`. For example, if we define

```
\def\try#1{\single#1*\endd}
```

then `\try{ X}` defines `\ans` to be `F`. (But `\try{␣␣}` defines `\ans` to be `T`—the second space never even gets read by TeX.)

It is now fairly easy to check whether an argument `#1` (which might *a priori* be an arbitrary token or even a group) is a comma. The basic idea is to define

```
\def\check#1,#2\endd
          {\empty#1\tricka\tricka\trickb}
```

and then define

```
\def\comma#1{\check#1,\endd}
```

so that `\comma{#1}` will define `\ans` to be `T` if `#1` is a comma, and `F` otherwise. This won't quite work for the following reasons:

(i) If `#1` is `{}` or `{{}}`, then `\comma{#1}` is `\comma{}` or `\comma{{}}`. This means that the `#1` appearing in `\check#1,\endd` is empty or `{}`, and thus the `#1` in `\empty#1\tricka\tricka\trickb` is empty.

(ii) If `#1` is a group `{,...}` that happens to begin with a comma, then `\comma{#1}` will define `\ans` to be `T`, whereas we want it to be `F` (this, admittedly, is a matter of taste).

So we will use `\emptygp` and `\single` to check on these possibilities:

```
\def\comma#1{\emptygp#1\endd
            \if T\ans{\gdef\ans{F}}
            \else{\single#1*\endd
                 \if F\ans{}
                 \else{\check#1,\endd}}}
```

Then

> `\comma{#1}`
> defines `\ans` to be `T` if `#1` is `,` or `{,}`, and `F` otherwise.

(The inability to distinguish between `,` and `{,}` is a minor problem that seems insurmountable.)

AMS-TeX needs many such checks, so they are all made in terms of one generalized check. For example, `\comma` is actually defined by

```
\def\comma#1{\compare*,{#1}}
```

where `\compare` is defined as

```
\def\compare*#1#2{\def\check##1#1##2\endd
           {\empty##1\tricka\tricka\trickb}
    \emptygp#2\endd
    \if T\ans{\gdef\ans{F}}
    \else{\single#2*\endd
        \if F\ans{}
        \else{\check#2#1\endd}}}
```

The `*` was made part of the syntax for `\compare` to allow `\def\space#1{\compare*␣{#1}}`.

## III. Saving Braces

We have just seen that there can sometimes be problems when braces are removed from the argument of a control sequence. Actually, the problem can be much more critical. For example, the AMS-TeX control sequence `\dots#1` first examines `#1` to determine what sort of dots and spacing are needed, and then produces these dots, followed by `#1` (and the remaining input). The removal of braces would be a minor annoyance if `#1` were something like `{+}`, where the braces are meant to make the `+` into a `\mathord` (something that AMS-TeX users aren't supposed to know about anyway). But it could be a major catastrophe if `#1` were something like `{a\frac b}`. To handle such problems we define

```
\def\braced#1{\empty
     #1\tricka\tricka\trickb
   \if T\ans{\gdef\Braced{{#1}}}
   \else{\single#1*\endd
        \if F\ans{\gdef\Braced{{#1}}}
        \else{\gdef\Braced{#1}}}}
```

In other words, `\braced` puts back a pair of braces if `#1` is `{}` or a group with more than one token or group in it. Thus, `\braced{#1}` defines `\Braced` to be `#1` except when `#1` is `{⟨token⟩}` or `{{...}}`, in which case the outer set of braces is removed. So, aside from the unavoidable `{⟨token⟩}` case, `\Braced` has enough braces to give the same result as `#1`.

## IV. Recursions

There are several ways of handling recursions, all of which are used at some point in AMS-TeX.

(a) Suppose that we want to define `\qms #1` so that

```
\qms 1     is ?
\qms 2     is ??
\qms 3     is ???
    ......
\qms {10}  is ??????????
    etc.
```

We can define

```
\def\qms#1{\setcount1 #1
     \def\string
         {\ifpos1{\advcount1 by -1
               \gdef\newstring{?\string}}
          \else{\gdef\newstring{}}
        \newstring}  %end of \def\string
     \string}
```

This only appears to violate the rule not to define a control sequence in terms of itself: An occurrence of `\string` may produce `\gdef\newstring{?\string}`, but TeX will simply record this definition, and not try to expand out the `\string` that occurs in it until `\newstring` is expanded, at which time an `\if` test is made, which produces a new `\gdef`.

`\newstring` should be defined as `?\string` rather than as `\string?` to keep TeX's internal "input stack" from growing unboundedly.

(b) Suppose that we have some input of the form

$\langle$`string`$_1\rangle$,$\langle$`string`$_2\rangle$, ... ,$\langle$`string`$_n\rangle$

with strings separated by some character, like a comma, and we want the control sequence `\operate` to perform some operation on each string. For example, we might want to replace each $\langle$`string`$_i\rangle$ by A$\langle$`string`$_i\rangle$Z, so that

`\operate{`$\langle$`string`$_1\rangle$,$\langle$`string`$_2\rangle$, ... ,$\langle$`string`$_n\rangle$`}`

will produce

A$\langle$`string`$_1\rangle$ZA$\langle$`string`$_2\rangle$Z...A$\langle$`string`$_n\rangle$Z

(We might also want to consider the case where there are no separators, so that an A and a Z will be inserted before and after each token or group.) We will use the token `\marker` as a "marker" to tell us when our recursion is over, so we define

```
\def\ismarker#1{\compare*\marker{#1}}
```

Now the basic idea is to define

```
\def\op#1,#2{\ismarker{#2}
             \if T\ans{A#1Z\gdef\nextop{}}
             \else{A#1Z\gdef\nextop{\op#2}}
             \nextop}
\def\operate#1{\op#1,\marker}
```

(omitting the commas in these definitions for the case of no separators).

Unfortunately this won't work, because there are problems concerned with the removal of braces. Each time `\op#1,#2` is used, argument `#2` is the first token or *group* following the comma, and if it is a group the braces will be removed. The removal of braces again causes problems if `#1` is something like `{a\frac b}`, and also if `#2` is something like `{(a,b)}`, where the braces are meant to "hide" the comma. We could use `\braced` here, but it isn't quite foolproof, since `#2` might be a "hidden" comma `{,}`, which `\braced` can't distinguish from an ordinary comma. Moreover, `\braced` can't help us with argument `#1`. Although this argument is usually a sequence, terminated by a comma, it just might happen to be a single group followed by a comma, and there is no way of distinguishing between these possibilities once argument `#1` has been read.

In the cases where AMS-TeX uses a recursive scheme of this sort, the particular circumstances, or simple tricks, usually circumvent these problems. The following definition illustrates a general scheme that will always work:

```
\def\kill#1{}
\def\op#1,#2\endd{\ismarker{#2}
       \if T\ans{A\kill#1Z\gdef\nextop{}}
       \else{A\kill#1Z\gdef\nextop
                            {\op*#2\endd}}
       \nextop}
\def\operate#1{\op*#1,\marker\endd}
```

Notice that each time `\op#1,#2\endd` is used, argument `#1` now begins with `*` (which is removed by `\kill`), so it can't possibly be a group. And argument `#2` is always the remaining input, terminated by `\marker`, so it can't be a group either.

(c) A recursive procedure can be used to count the number of commas in a string:

```
\def\cm#1,#2\endd{\ismarker{#2}
          \if T\ans{\gdef\nextcm{}}
          \else{\advcount1
                \gdef\nextcm{\cm#2\endd}}
          \nextcm}
\def\countcommas#1{\setcount1 0
                   \cm#1,\marker\endd}
```

Then

```
        \countcommas{#1}
        makes the value of \count1 be the
        number of commas in #1.
```

The \endd trick is used to handle "hidden" commas, but the * trick isn't needed, since we don't care what \cm does to #1.

(d) If we do \countcommas{#1}, then \ifpos1 will tell us whether #1 contains at least one comma. But it is preferable to use the following scheme, which doesn't involve any counters, and which stops as soon as the first comma is found:

```
\def\cm#1,#2{\ismarker{#2}
          \if T\ans{\gdef\nextcm{}}
          \else{\gdef\Hascomma{T}
                \gdef\nextcm##1\marker{}}
          \nextcm}
\def\hascomma#1{\gdef\Hascomma{F}
                \cm#1,\marker}
```

(e) Suppose we want to perform the operation in part (b) on some input of the form

$\langle\texttt{string}_1\rangle\texttt{\textbackslash\textbackslash}\langle\texttt{string}_2\rangle\texttt{\textbackslash\textbackslash}\ldots\texttt{\textbackslash\textbackslash}\langle\texttt{string}_n\rangle$

where the separator is the control sequence \\ (which is never used in isolation, and is initially defined by \def\\{}). We could use exactly the same scheme, replacing \def\op#1,#2\endd by \def\op#1\\#2\endd. But we can also take advantage of the fact that the separator is a control sequence to obtain a definition that is both more elegant and more efficient:

```
  \def\op#1\\{A\kill#1Z\\}
  \def\operate#1{\def\\{\op*}
                 \op*#1\def\op{\kill}\\
```

The \def\op{} needs to be replaced by \gdef\op{} if \op puts things inside braces; in this case, the original definition of \op should be made part of the definition of \operate.

There might appear to be possible confusion if some $\langle\texttt{string}_i\rangle$ contains \\ within a group {...\\...}. In $\mathcal{A}\kern-.1667em\lower.5ex\hbox{$\mathcal{M}$}\kern-.125em\mathcal{S}$-TEX this occurs only in constructions like

```
        {\align...\\...\endalign}
```

where \\ is temporarily re-defined anyway.

**V. Searching For Strings**

TEX's method of determining where an argument in a definition ends has the following peculiar feature. Suppose we define

```
        \def\cs#1ab#2{...}
```

Then the first argument is the smallest (possibly empty) token or group that is followed by a, *not* the smallest group that is followed by ab. So the input

```
        \cs xayabc
```

gives the error message

```
! Use of \cs does not match its definition.
```

So if we want to know whether ab occurs in some string we can't simply replace the comma by ab in the method of part IV(d), because an a might occur alone. Instead we have to do something like the following:

```
\def\isb#1{\compare*b{#1}}
\def\finda#1a#2#3\endd{\ismarker{#2}
     \if T\ans{\gdef\nextfinda{}}
     \else{\isb{#2}
          \if T\ans{\gdef\Hasab{T}
                   \gdef\nextfinda{}}
          \else{\gdef\nextfinda
                   {\finda#2#3\endd}}}
     \nextfinda}
\def\hasab#1{\gdef\Hasab{F}
            \finda#1a\marker\endd}
```

* * * * * * * * * * *

## Problems

* * * * * * * * * * *

The first formatting problems posed in this column come from the videotaped TEXarcana Class taught by Don Knuth last March. Solutions will be presented in the next issue. Readers with working TEX systems are encouraged to attempt solutions to these problems, in order to better appreciate the problems and their solutions.

Lynne A. Price

**Problem no. 1:**

*Type:*

```
\vskip 12pt
\noindent\hide{--}Allan Temko

\vskip 2pt
\noindent Architecture Critic
```

*To get:*

*–Allan Temko*
*Architecture Critic*

**Problem no. 2:**

*Type:*

```
\fancy Senator and Mrs.\Stanford had reserved to themselves control of the
University's affairs during their lifetimes, including the parceling
out of ``all the money that could be wisely used.'' Mrs.\Stanford had remained in
her husband's shadow---on opening day she could not bring herself to deliver
the short speech she had written out.  But following the death of the Senator
she, at age 65, took on full responsibility for the University with
unsuspected strength.
```

*To get:*

Senator and Mrs. Stanford had reserved to themselves control of the University's affairs during their lifetimes, including the parceling out of "all the money that could be wisely used." Mrs. Stanford had remained in her husband's shadow—on opening day she could not bring herself to deliver the short speech she had written out. But following the death of the Senator she, at age 65, took on full responsibility for the University with unsuspected strength.

**Problem no. 3:**

*Type:*

```
\hsize 25em
\noindent This is a case where the name and address fit in nicely
with the review.\signed{A. Reviewer}{Ann Arbor, Mich.}

\vskip 8pt
\noindent But sometimes an extra line must be added.\signed{N. Bourbaki}{Paris}
```

*To get:*

This is a case where the name and address fit in nicely
with the review. *A. Reviewer* (Ann Arbor, Mich.)

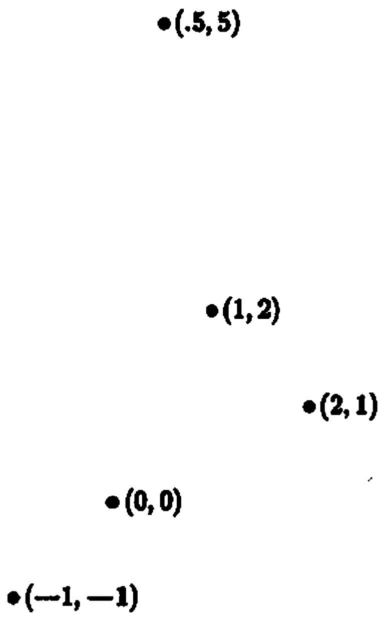
But sometimes an extra line must be added.
*N. Bourbaki* (Paris)

**Problem no. 4:**

*Type:*

```
\point 0 0
\point 1 2
\point 2 1
\point .5 5
\point -1 -1
```

*To get:*



**Problem no. 5:**

*Type:*

```
\hsize 20em
End of a paragraph.\par
\rightjustifythefollowing
This is the first line
{\it This is the second line.}
{\sl The third.}
{\bf The last.}
\endrightjustify
Beginning of another paragraph.
```

*To get:*

End of a paragraph.

This is the first line.
*This is the second line.*
*The third.*
**The last.**

Beginning of another paragraph.

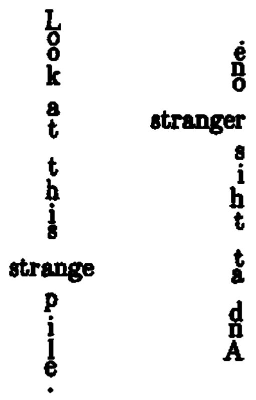
**Problem no. 6:**

*Type:*

```
How do you do this?
$$\lineskip 2pt
\baselineskip 1.3ex
\vcenter{\halign{\hfil#\hfil\cr
\linedown{Look at this {strange} pile.}}}\qquad
\vcenter{\halign{\hfil#\cr
\lineup{And at this {stranger} one.}}}$$
```

*To get:*

How do you do this?



**Balancing Columns of Text and Translation**

I would like to typeset translations in parallel with original texts using TEX. Perhaps there is a TEXnician who can solve a formatting problem concerning this type of typesetting.

It should be possible to recalculate the size of blocks taken by each language until both languages end on the same word at the bottom of their block. Suppose that an initial estimate is made such that language A consumes 50% of the page and language B consumes 40% of the page. Ten percent of the page is taken for margins. When language A is at the bottom of its block, language B has only consumed 90% of its block. By making the column of language A approximately 5% wider and the column of language B 5% narrower, the last word of both languages will more nearly come to the end of the block. Is there an easy macro that will do this in TEX?

Johnny Stovall

* * * * * * * * * * *

**Input-Dependent Macro Redefinition**

I would like a way of combining various (non-successive) occurrences of certain types of input as the values of a macro. For example, initially we might define `\list#1{}`. Then an occurrence of `\data{...}` in the input file should redefine `\list` so that `\list 1` is ..., while `\list 2`, `\list 3`, etc. are empty. Another occurrence of `\data{***}` sometime later should redefine `\list` so that `\list 1` is ..., `\list 2` is ***, `\list 3` is empty, etc., etc.

Does anyone know how to do this?

Michael Spivak

* * * * * * * * * * *

## Letters

* * * * * * * * * * *

Dear TUG Members:

It was mentioned at our last meeting that TUGboat has yet to receive any "letter to the editor" submissions. I would like to help rectify that lack by stating my worries about the effectiveness of TUG. The Steering Committee is extremely reluctant to adopt any formal structure or bylaws. We certainly want to avoid unnecessary regulation and such looseness is fine as long as it does not prevent the committee from functioning. We do want to impose certain constraints—I believe the Steering Committee did decide, for example, that, while each of its members is free to define his own rôle, site coordinators should not discourage relevant telephone calls.

The committee members are very aware that opinions differ and are reluctant to take action that might impose their views on the group as a whole. I fear that this admirable attitude, in conjunction with an informal structure, may result in an ineffective Users Group. As a case in point, Sam Whidden mentioned in May that the Steering Committee had decided against assigning the maintenance and distribution of TEX to a software house. There was considerable discussion of this point in January. Bob Morris eloquently argued about the dangers to university users of such an approach. I was not aware, however, that Bob had succeeded in convincing the committee as a whole. I had supposed that the finance committee would have prepared alternate proposals before this last meeting, that there would have been more discussion, and that a final decision would have been based on a vote. Certainly we cannot continue to abandon proposals simply because they engender heated discussion.

The same attitude emerged in the schedule for the "Implementors' Workshop". The program for the entire second day of the two-day meeting was left unplanned in order to allow attendees to raise issues of their own interest. With the limited amount of time available, the breadth of the information to be covered, and the number of opinions to be solicited, it might have been better for someone to have taken the responsibility of making the decisions ahead of time. The intent of the meeting was to provide demonstrations of output devices and discussions of TEX implementations on various architectures. These topics were postponed until the end of an intense conference. While the other material was of unquestionable value, it was of most interest to users who currently have access to TEX and to individuals considering acquiring TEX rather than to those who have decided to install TEX but have not yet succeeded in doing so. It is ironic that Richard Palais pointed out that it has been over a year since a general meeting of all TUG members—surely, had it been so advertised, this meeting could have been one. It is also ironic that Phil Sherrod suggested small workshops hosted by assorted TEX sites to describe their own installations. Such a suggestion indicates that this meeting did not fulfill its intended purpose.

The May 14th Steering Committee meeting was open to the membership as a whole. The Steering

Committee certainly wants its actions to be visible, it wants to solicit the opinions of others, and to encourage new volunteers. However, by the time all participants in such a large group have voiced their opinions, it is impossible for decisions to be reached. We need an effective decision-making process.

The Steering Committee has also proposed raising individual membership fees and establishing institutional memberships. This action has been delayed until TUG determines the services it will offer in exchange for such funds. A current situation illustrates both the need for some formal organization and the need to raise money. The ANSI X3J6 committee on text processing language standards is meeting June 22–26. Experienced users of two other mathematical typesetting systems have been invited to present the software with which they are familiar. This ANSI committee has asked that a TEX user also participate. Although it is likely I will join X3J6, I am unable to attend the upcoming meeting. Mike Spivak has volunteered to substitute for me, but does not have institutional support for his travel expenses. The officers of the TUG Steering Committee strongly feel the Users Group should support this activity. However, our treasury is empty and it is not clear who can authorize such expenses.

The cure for this chaos is more work by the Steering Committee. Sub-committees should meet (even electronically or by telephone) between general meetings. Someone must accept the responsibility of organizer and must be willing to make decisions, even if they are temporary decisions later vetoed by vote of the entire membership. I am as guilty as anyone else of neglecting my Steering Committee responsibilities except during meetings and the few days before TUGboat submission deadlines. I, for one, will attempt to be more active in the coming months.

Sincerely yours,
Lynne A. Price

* * * * * * * * * * * *

## Dreamboat

* * * * * * * * * * * *

*Send Submissions to:*
Lynne A. Price
Calma R&D
212 Gibraltar Dr.
Sunnyvale, CA 94086

One refreshing quality of the TEX user community, and particularly of the system's creator, is that TEX is viewed, in fact intended, to be the ancestor of an evolving family of document formatters rather than as a static piece of software that will be used for decades. DREAMBOAT is a feature of TUGboat where users can describe (in whatever detail) capabilities they would like to see implemented in some successor system.

A brief "Son of TEX" session was held at Stanford in May. Extensions of immediate interest include applications to non-mathematical documents, even those printed in languages other than English. The foreign language application requires replacement of the English-based hyphenation module. For Hebrew and Arabic, right-to-left formatting would be convenient. There is also current interest in interfacing a general graphics capability with TEX. As described in the last TUGboat, Vanderbilt University has modified the Versatec spooler to allow output of plot files created in a format compatible with their Zeta pen plotter. They intend to modify TEX so that plot files can be merged with TEX output. Other installations are working on graphics extensions.

TEX's user interface, particularly the input language and error messages, was also discussed, as an area to be improved in the less immediate future. One specific point mentioned was the difficulty of identifying which spaces and carriage returns are significant. Macro languages in general were criticized. The controversial suggestion was made that future systems be more like programming languages. Joe Weening, a Stanford student, described his work on a TEX derivative called LaTEX, which is a hybrid of TEX and Lisp. In LaTEX, one can escape from TEX into Lisp, to do complex computations or text manipulations which are difficult or impossible to do in TEX.

Other topics included page markup and an interactive ("what you see is what you get") version of TEX. There was some discussion of a feature that enabled users to tell where on a page material was being placed. David Fuchs pointed out that such a feature is incompatible with TEX's algorithm for determining page breaks.

* * * * * * * * * * * *

**Contents**

*July 1981*

***** Continued *****

**Contents — Continued**