



# TUGBOAT

THE TEX USERS GROUP NEWS LETTER

february 1981

VOLUME 2. NUMBER 1.  
PROVIDENCE RHODE ISLAND

USA

## **TUG Steering Committee**

**Donald Knuth, Grand Wizard of TeX-arcana**  
**Richard Palais, Chairman of the Steering Committee**  
**Robert Morris, Secretary and Wizard of Macros**  
**Samuel Whidden, Treasurer**  
**Robert Welland, TUGboat Editor**  
**Luis Trabb-Pardo, Vice Grand Wizard**  
**Ignacio Zabala, Wizard of TeX-in-Pascal**  
**David Fuchs, Wizard of I/O and SAIL**  
**Barbara Beeton, Wizard of Format Modules**  
**Nicholas Allen, Wizard of TOPS-10**  
**Arnold Pizer, Wizard of TOPS-10**  
**Phil Sherrod, Wizard of TOPS-10, Coordinator for DEC10**  
**Patrick Milligan, Wizard of TOPS-20, Coordinator for DEC20**  
**Michael Spivak, Wizard of AMS-TeX**  
**Richard Zippel, Wizard of ITS**  
**Hermann Zapf, Wizard of Fonts**  
**Charles Howerton, Liaison with the  
National Bureau of Standards**  
**Richard Friday, Liaison with Digital Equipment Corporation**  
**Michael Bennett, Foreign Distribution of TeX**

### **Site Coordinators**

**Eagle Berns, Coordinator for IBM 370**  
**Thea Hodge, Coordinator for CDC Cyber**  
**Scott McCourt, Coordinator for Burroughs**  
**Monte Nichols, Coordinator for VAX**  
**Ralph Stromquist, Coordinator for Univac 1100**

Very few composers are fond of algebra.

Thomas MacKellar  
*The American Printer:*  
*A Manual of Typography,*  
Philadelphia,  
Mackellar, Smiths & Jordan, 1871.

# TUGBOAT

THE T<sub>E</sub>X USERS GROUP NEWSLETTER  
EDITOR ROBERT WELLAND

VOLUME 2, NUMBER 1  
PROVIDENCE

RHODE ISLAND

FEBRUARY 1981  
U.S.A.

## ADDRESSES OF TUGboat AUTHORS

6 February 1981

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

**BERNS, Eagle**  
 Polya 207  
 Stanford Center for Information  
 Processing  
 Stanford University  
 Stanford, CA 94305

**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

**ECK, David**  
 Department of Mathematics  
 Dartmouth College  
 Hanover, NH 03755

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

**HIRST, Graeme**  
 Department of Computer Science  
 Brown University  
 Providence, RI 02912  
 401-863-3300

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

**INCERPI, Janet**  
 Department of Computer Science  
 Brown University  
 Box 1910  
 Providence, RI 02912  
 401-863-3342

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

**KIM, Scott**  
 Stanford University  
 P.O. Box 9414  
 Stanford, CA 94305  
 415-329-9081

**LAWSON, Charles L.**  
 Jet Propulsion Lab  
 MS 125/128  
 4800 Oak Grove Drive  
 Pasadena, CA 91103  
 213-354-4321

**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

**PALAIS, Richard S.**  
 Department of Mathematics  
 Brandeis University  
 Waltham, MA 02154  
 617-647-2667

**PRICE, Lynne A.**  
 Bell Northern Research, Inc.  
 685A Middlefield Road  
 Mountain View, CA 94043  
 415-969-9170

**SHERROD, Phil**  
 Box 1577, Station B  
 Vanderbilt University  
 Nashville, TN 37235  
 615-322-7311, ext. 2951

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

**TOTLAND, Helge**  
 Computas A/S  
 Box 310  
 N-1322 Høvik, Norway

**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

**WRIGHT, Alan**  
 Box 1577, Station B  
 Vanderbilt University  
 Nashville, TN 37235  
 615-322-7311

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

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, R.I. 02940. Annual dues for individual members of TUG are \$10.00; 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, R.I. 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, Ill. 60201; items submitted on magnetic tape should be addressed to Barbara Beeton or Barry Doherty, American Mathematical Society, P.O. Box 6248, Providence, R.I. 02940.

Submissions to TUGboat are not refereed. Some are minimally edited, and some are reproduced directly from copy submitted by the authors. Any questions regarding the content or accuracy of particular items should be directed to the authors.

Editor's note: This issue is especially large because all items received by press time have been included. The Editor believes that information should be relayed to the TUG membership as soon as possible, so nothing is being held over for another issue.

\* \* \* \* \*

### General Delivery

\* \* \* \* \*

### SITE COORDINATORS

Robert Welland

The following people, who are bringing up  $\TeX$  on machines at their institutions, have agreed to be site coordinators. Their primary responsibility is to get  $\TeX$  running and when this is done to write up a report for TUGboat. They have also agreed to answer a limited number of questions; they offer this help for free and the time to do so comes from very busy schedules. If you are not involved with bringing up  $\TeX$ , please wait for the site reports to appear in TUGboat. Hopefully, they will answer most questions and make manageable the burden the site coordinators will have to bear.

If you are bringing up  $\TeX$  on one of the following machines please inform the appropriate person; otherwise send the information to the TUGboat editor, Robert Welland.

<i>Machine</i>	<i>Coordinator</i>
Burroughs B88000	Scott McCourt Burroughs Corp. B.C.S. Project Corporate Drive, Commerce Park Danbury, CT 06810 203-794-0191 ext 515
CDC Cyber	Thea Hodge University Computing Center 208 Union St. S.E. University of Minnesota Minneapolis MN 55455 612-373-4599
DEC10 running under TOPS-10	Phil Sherrad Box 1577 Station B Vanderbilt University Nashville, TN 37235 615-322-7311 ext 2951
DEC20 running under TOPS-20	Patrick Milligan Bell Northern Research Inc. 685 A Middlefield Rd. Mountain View, CA 94043 415-969-9170 ext 2837
IBM 370	Eagle Berns Polys 207 Stanford Center for Information Processing Stanford University Stanford, CA 94305 415-497-4382
Univac 1100/82	Ralph Stromquist Academic Computing Center University of Wisconsin 1210 W. Dayton St. Madison, WI 53706 608-262-8821

VAX

Monte Nichols  
Sandia National Laboratories  
Livermore, CA 94550  
415-422-2706

\* \* \* \* \*

### CHAIRMAN'S REPORT

Richard S. Palais

The Steering Committee of TUG met for most of the day on January 9 at the San Francisco Hilton. Much of what transpired at that meeting is reported on elsewhere in this newsletter. I would like to concentrate here on two aspects of the discussions. The first of these is a continuing strong division of opinion on the question of  $\TeX$  "maintenance", a matter that many will remember already evoked considerable debate at the first TUG meeting. The disagreement is over which of two goals, both obviously desirable, should take precedence. One goal is that  $\TeX$  should remain as "free" as possible and the other is that  $\TeX$  should be as carefully and professionally maintained as possible. At one extreme are the large "production" users who would like rapid and dependable advice and help with all their software problems. For them  $\TeX$  will be one module in a complex system. They have deadlines to meet that require that all these modules work, and they are used to, willing, and able to pay up to several thousand dollars per year to have real or imagined bugs exorcised on the spot and have their software tailored, tuned, and customized for them. At the other extreme is the single, small user with little or no money to spend but competent, willing and able to invest his time and effort in "hacking  $\TeX$ " for himself. Complicating matters is a third and perhaps over-riding goal, the need to assure that there remains a single, standard " $\TeX$ ", compatible across many machine architectures and output devices. Fortunately these goals and the constituencies supporting them are not really conflicting, but rather orthogonal. With care and compromise there does not seem to me to be any serious reason why the various categories of  $\TeX$  users cannot all have their needs met. But it is clear that to avoid nonproductive conflicts and polarizations everyone in TUG will have to keep in mind that the TUG membership is anything but homogeneous and several different options will frequently have to be provided to satisfy all the different classes of  $\TeX$  users.

The second matter I would like to discuss is the Steering Committee's decision to call for a  $\TeX$  Implementation Workshop at Stanford in the middle

of May. This will be a two day meeting. One day will be a  $\text{\TeX}$  demonstration day, open to all present and prospective TUG members. This is meant to give an opportunity to become familiar with the various components of the  $\text{\TeX}$  system, and in particular with the different output device options. (At present at Stanford it is possible to have a  $\text{\TeX}$ -produced DVI file output on any one of a Xerox XGP or a Versatec electrostatic printer/plotter, an Alphatype CRS typesetter, or a Canon or Xerox (Dover) laser printer.) The other day is aimed primarily at those actively or prospectively engaged in the implementation of  $\text{\TeX}$  systems and the goal is to maximize the amount of help and information these people can exchange with each other and with the central  $\text{\TeX}$  team at Stanford. The ultimate goal of this implementation project is to be able to supply "off the shelf" to anyone desiring it all the components of a completely working  $\text{\TeX}$  system. Let us consider for the moment what these components are:

(A)  $\text{\TeX}$ -in-Pascal.

- (1) System independent part.
- (2) System dependent part.

(B) Font files.

- (1) Font information files (device independent).
- (2) "Character shape" files.

(C) DVI-to-hardcopy back end.

- (1) Output device hardware interfaces.
- (2) Software for output device interfaces.
- (3) Queuers, spoolers, device drivers for output devices.
- (4) Character shape file "pipeline" from host disk system.

Part (A) (together with B1) is what is necessary to produce DVI output files from a valid  $\text{\TeX}$  input file. Now A1 has long been complete, and A2 is either completed or nearing completion for a wide spectrum of host machines of different manufacture, architecture and operating systems (DEC TOPS-10, TOPS-20, VAX VMS, VAX UNIX; IBM 360/370; CDC Cyber; Univac 1100). I think that we can look forward with some confidence to the May meeting as marking the virtual completion of this first phase of  $\text{\TeX}$  implementation. Now as for part B, the creation of a basic font library, that too is essentially complete. The whole family of CM fonts (and others besides) now exist as **METAFONT** programs. Recall from this column in the first number of the TUG newsletter that **METAFONT** not only creates the device independent font information file (containing the size, spacing, kerning, and ligature information needed by  $\text{\TeX}$  to create a DVI

file), but also, once a simple interface program is written for a given output device, **METAFONT** will create the character shape files, in the form of raster patterns stored as, say, matrices of zeros and ones. Such **METAFONT** interfaces have now been written for over a half dozen output devices, running from the super high resolution (5300 dot/inch) Alphatype CRS to the low resolution (128 dot/inch) Florida Data impact printer.

So what is rapidly approaching is the final phase of the  $\text{\TeX}$  implementation program, the creation of the back-end systems which for a given host mainframe and output device will, from the DVI file and character shape files, produce the hardcopy output. Now as David Fuchs has remarked, life would be quite easy if output devices had built into them enough disk-type storage to handle all the character shape files for sixty-four fonts of 128 characters each and enough logic to process the DVI files into raster scan lines. One would still have the (rather trivial) job of writing for each operating system spoolers and queuing programs to send DVI files over a serial line to this ideal output device in an orderly fashion, but one would be able to avoid a host of other small headaches that real world output devices force one to deal with. Since, in fact, output devices usually have no usable general purpose microcomputer built in, one must interface the host computer to the output device via a microcomputer able to speak to both. Also, since all the character shape files that must be accessible to process a complex DVI file can in principle run in the megabyte range, economic considerations mandate that with current technology these files must be kept on the host computer disk memory, and then downloaded as necessary to a small floppy disk system associated to the microcomputer interface over the (for simplicity) serial line joining it to the host computer. Now, designing the hardware interface from off the shelf items and writing the software to make it all go is not a major project for an expert systems programmer who understands the format of DVI files and character shape files and knows how to communicate scan lines to the output device. Perhaps a month or two of hard work will suffice. What is frustrating is that these systems are extremely sensitive to small differences in the various protocols of operating systems and output devices, so if there are  $m$  of the former and  $n$  of the latter one could easily end up doing this same work  $mn$  times. Of course common sense tells us that good planning should be able to reduce this to more like  $m + n$  times. (For example, David Fuchs could probably interface one more output device to TOPS-20 in under a week.)

A major reason for the May meeting is to reduce as much as possible unnecessary duplication of effort in this final part of the program to make  $\text{\TeX}$  generally available.

\* \* \* \* \*

## REPORT ON THE TUG STEERING COMMITTEE MEETING

The TUG Steering Committee and several observers met at the San Francisco Hilton on January 9, 1980. Below is reported the gist of that meeting as recovered from tape recordings and my notes. Since several topics re-emerged throughout the meeting, I have not reported in any order related to that of the meeting.

Robert Morris

### ATTENDANCE

The following attended:

Barbara Beeton, AMS Providence  
 Max Díaz, Stanford  
 Barry Doherty, AMS Providence  
 David Fuchs, Stanford  
 Ellen Heiser, AMS Providence  
 Don Knuth, Stanford  
 Leslie Lamport, SRI  
 William LeVeque, AMS Providence  
 Patrick Milligan, Bell Northern Research  
 Robert Morris, UMASS/Boston  
 Evon Motiska, Stanford  
 Monte Nichols, Sandia Labs  
 Richard Palais, Brandeis  
 Lynne Price, Bell Northern Research  
 David Rogers, University of Michigan  
 J. L. Selfridge, Math Reviews  
 Phil Sherrod, Vanderbilt  
 Michael Spivak, Decatur, Ga.  
 Rilla Thedford, Math Reviews  
 Luis Trabb-Pardo, Stanford  
 Bob Welland, Northwestern  
 Sam Whidden, AMS Providence  
 W. B. Woolf, Math Reviews  
 Ignacio Zabala, Stanford

### TREASURER'S REPORT

Sam Whidden gave the treasurer's report, attached. The cost of producing the first newsletter exceeded the treasury by \$419. It is estimated that an additional \$3600 is needed for two issues in 1981.

### MISCELLANY

Pat Milligan and Lynn Price of Bell Northern Research have been extensively developing macros in-house and have had successes making overhead slides, Hebrew, and special graphics.

### AMS- $\text{\TeX}$

Mike Spivak reported that version —1 works as indicated in *The Joy of  $\text{\TeX}$* , where not-yet-implemented features are indicated in handwritten marginalia. The *Joy* was not processed by AMS- $\text{\TeX}$ , which will now enter its field testing phase as people begin to use it. An order form for a tape is included with the manual, for sale at AMS headquarters. The first finder of each manual misprint will receive a \$1 bounty, and the first finder of each AMS- $\text{\TeX}$  bug a \$5 bounty. Bounty may be claimed by writing Mike at the address in *The Joy of  $\text{\TeX}$* . The *Joy* was produced on the Providence Alphatype, which is exhibiting some backlash problems resulting in distortion of some vertical lines.

AMS- $\text{\TeX}$  has too many macros for easy use in the SAIL  $\text{\TeX}$  running now at Providence. The Pascal version is not expected to have the size limitations which caused the problems, which in any case can be changed by recompiling  $\text{\TeX}$  with bigger values of hashsize. The Pascal version will provide 3 bits more address space for internal memory than the SAIL version and such problems will not be serious. A similar problem with memsize appears when setting multi-column output. These should also disappear in the Pascal version. Making AMS- $\text{\TeX}$  macros more efficient will help, which Mike will do this week.

### MAINTENANCE

The administrative burden of maintaining  $\text{\TeX}$  has become too large for Stanford to support on the informal basis they do. Throughout the meeting at various times debate raged on the appropriate mechanism for maintenance. Since this is inextricable from membership fees, a Finance Committee was formed to recommend a maintenance policy, to recommend a membership fee policy, and to explore sources of support, e.g. foundations. This consists of Sam Whidden, Luis Trabb-Pardo, chairman, Bob Morris, Pat Milligan and Monte Nichols. It seemed that everyone agreed that TUG would run a  $\text{\TeX}$  switchboard whereby someone would be paid to tell callers who can answer their questions. Luis and the Stanford people are spending too much time doing this and answering the questions. The switchboard could also refer people to an up-to-date list of consultants for hire.

The (unsettled) argument about maintenance varied between two positions: (a) Some organization with either an explicit financial interest or an in-house  $\text{\TeX}$  support facility should maintain  $\text{\TeX}$  at TUG's expense. (b) There should be no financial burden whatsoever on the membership and no particular  $\text{\TeX}$  maintenance should be endorsed by the

Users Group (see separate articles in this newsletter). Knuth's intention is that the released Pascal TeX will be a single stable core (aside from the system dependent module) which can be uniformly maintained for all versions and should not have any supported enhancements.

#### TUG MEETING

No general TUG meeting will be called until the Pascal TeX is released. However, an Implementor's Workshop has been called for May 14-15 at Stanford. This will be coupled with a TeX open house comprising demonstrations for people who don't know what TeX is. Details are elsewhere in this issue.

#### PASCAL TeX

A detailed report of each architecture appears elsewhere in this issue. Knuth expects to read the Pascal version's code this spring very carefully before its public release. When released, it will be frozen with no enhancements or changes perhaps aside from bug fixes. At present, the only fully operational Pascal version is the TOPS-20 implementation. One problem is that implementors have also to get their output hardware working in order to see output. In general interfacing output devices is proving a greater share of the implementation efforts than people imagined, but has nothing directly to do with bringing up Pascal versions.

#### EDUCATION

Knuth is making a video tape to teach TeX to users. He intends for this tape ultimately to be available through TUG.

#### OUTPUT DEVICES

Phil Sherrod: It took a week to get the SAIL version working on TOPS-10, but nearly a year to get all the output device interfaces conveniently working (e.g. spoolers) although only a month to get something up. Finding the right hardware interfacing was lengthy and mysterious. TUG should maintain descriptions of what kind of hardware to buy and what the software interfaces involve.

Phil will write an article describing the tribulations of output device implementors.

There was a belief expressed that the output device vendors would have to be involved in output interfaces. Cooperation greater than that already provided by manufacturers of existing devices will be needed for wide applicability. Luis Trabb-Pardo expressed the belief that more intelligence needs to be provided in devices in order to relieve burden on the host, which will allow less system dependent software. The problem is that Xerographic printers are or soon will be selling for about \$3000 for the

printing engine. The interface will cost about the same, bringing the OEM cost to about \$6-7,000. The end-user prices will be around \$20,000 for complete printing systems. These will have sufficient intelligence to take DVI files more or less directly. A similar arrangement based on electrostatic printers should sell for around \$10,000 end-user system price.

Math. Reviews has interfaced a Florida Data dot matrix printer with the same mechanism (a \$3500 one-board Z80 system) to drive it in graphics mode. The device has 128 dots/inch and might be suitable for very rough copy. It is doing TeX output at about 30 seconds per page, which is a little slower than the electrostatic devices.

Varian, Versatec, Dover, and Alphatypes are working at several sites. The Dover does not accept DVI files and is in any case not commercially available.

\* \* \* \* \*

### 1980 TUG TREASURER'S REPORT

Samuel B. Whidden

During 1980 the first issue of TUGboat appeared. The costs associated with its printing and distribution amounted to \$1,719. (Not included in this figure are costs for services provided by AMS professional staff.) As of December 31, 1980, 130 membership applications have been received for a total income of \$1,300.

Income:	Membership		\$1,300
Expenses:	Printing	\$1,232	
	Postage	371	
	Mailing/Labor	116	1,719
Balance (as of 12/31/80)			(\$ 419)

Based on the costs for 1980, it is anticipated that direct costs associated with the production and distribution of two issues of TUGboat during 1981 will be approximately \$3,600, with the AMS continuing to contribute the services of its professional staff.

Respectfully submitted,  
Samuel B. Whidden, Treasurer  
January 5, 1981

(Note: As of 2/6/81 a total of 258 paid membership applications had been received.)

\* \* \* \* \*

### INFORMAL TUG SESSION

Robert A. Morris

On the afternoon of January 9, various TUG Steering Committee members and other interested



people met at the Hilton for informal discussion. Among the people I can remember were there were: Lynne Price and Pat Milligan (Bell Northern Research), Leslie Lamport (SRI), Luis Trabb-Pardo (Stanford), Rilla Thedford (Math Reviews), Arnie Pizer (Rochester). Possibly I have missed some. I have reported below some of the wishes, rumors, and reports from this meeting and other sources. Nothing is guaranteed accurate!

Lynne and Pat have extensive macro experience and have made slides, Hebrew, and are advocating  $\TeX$  as a standard for in-house technical documentation in their organisation.

Some desires: improved user interfaces. Lynne will start collecting complaints and suggestions.  $\text{AMS-}\TeX$  is an example.

Many people want a  $\TeX$  preprocessor which can run on small machines with which people can test the syntax of their  $\TeX$  input. (But Unidot has its C version running on an Onyx UNIX system with a Versatec printer and hopes for release soon. This could presumably be used even without an output device. It apparently is based on the SAIL version of  $\TeX$ . Will it be released with all the changes which end up in the Pascal version? It will be for sale.)

Luis: In principle, all that is needed for the use of an arbitrary printing engine is cooperation from the vendor in providing (a) Font Metrics for each font; (b) If the galley proof cost is high, will they provide proof mode encodings (e.g. at 200 dots/inch), not only thereby protecting their own font investment but allowing users to run proof mode versions of their fonts on a proof device? (c) Is the typesetter language available to people to write DVI-to-device drivers? (d) Is the manufacturer willing to include math fonts made by METAFONT?

A trick if your macro packages are too big to fit (which shouldn't happen very much in the Pascal versions): redefine as null macros which will not be used again. This will return the space to the memory manager.

Leslie Lamport: A trick to avoid un-matched brace syntax errors: When using a screen editor like EMACS, use a macro which creates matching braces with nothing between them except the cursor. The closing brace is then automatically there after the text entry is finished.

\* \* \* \* \*

**Editor's note:** The following two articles give divergent views on the subject of how the  $\TeX$  program is to be maintained in the future. Readers are invited to comment, and to make known their own views on the

subject before the next meeting of the TUG Steering Committee in May.

\* \* \* \* \*

## A POSITION ON $\TeX$ MAINTENANCE

Robert Morris, UMASS/Boston

There are at least two diametrically opposed maintenance/distribution models we can consider. The production user, like AMS or a commercial user, wants something like a fully supported  $\TeX$ . Such a user has a calculable, or at least identifiable, financial penalty which it incurs when the software it is using does not work. On the other hand is the university user facing little or no budget resources which it can devote to buying support, but also having no particular time constraints and having a pool of talent—its students—which can tweak non-working software. Since I am in the second community, I would like to argue in favor of TUG's involvement being closer to the second model, while facilitating the other class of users solving their support needs at their own expense.

I believe that few universities would benefit by paying \$1,000/year membership fee to TUG. Indeed, \$200 seems too much to me for an organisation which can wait weeks to get sick software fixed. Instead of contributing to extensive  $\TeX$  support, I suggest that there be designated distribution sites for each architecture, selected from some organization heavily using  $\TeX$  on that architecture. These sites would make standard release tapes at cost and would incorporate bug fixes at designated intervals (quarterly?). Their interest would lie principally in being the funnel for proposed bug fixes (which would often be proposed by the discoverer of the problem) and thereby having the first and widest perspective on maintenance. Presumably these would be organizations which are already maintaining  $\TeX$  in-house and thus have sufficient expertise to recognize whether a bug report is in fact a  $\TeX$  problem or a user problem.

The other side of this essentially un-supported  $\TeX$  is that users can tinker with  $\TeX$  and circulate their own "enhancements". On the one hand, this is contrary to desires that Knuth has expressed. On the other hand, it is bound to happen when sources are distributed, and I am not convinced it is bad. The most successful model of this kind of un-supported source distribution is the UNIX operating system. UNIX is distributed free to educational users with source licenses. Tapes are made at cost by the licensor, Bell Laboratories. Often, these

releases do not work on the precise configuration the licensee has and varying degrees of work are required to bring up the system. Alternatively, users often get copies of the system not from Bell Laboratories but from another site with the same or similar configurations. Many users modify their systems and/or install major modifications made by other sites. This process continued for 6-7 years throughout the life of "version 6 UNIX", the first version in wide circulation. All this experimentation led to two things: a bizarre proliferation of somewhat incompatible versions of UNIX and a substantial base of expertise about the system coupled with a great deal of experimentation toward modernizing the operating system. The result of the former was that UNIX came to be regarded as needing substantial systems programming expertise to keep it running (a false belief which did not take into account the simplicity of the operating system and the ease of dealing with code written in a high-level language). The result of the latter was that version 7 UNIX and that for the VAX have incorporated the results of these experiments and apparently produced a very contemporary and useful operating system which internally looks little like version 6, but to users is very similar. After all this tinkering, the resulting product seems to be useful not only in universities, but at high prices in commercial environments.

On the one hand, such a model seems incompatible with Knuth's position that T<sub>E</sub>X will be released in such a way that tinkering is un-needed. He prefers that people think of T<sub>E</sub>X as something which will not need enhancement, but rather will be the foundation of similar future developments which are not T<sub>E</sub>X but (hopefully) something better. On the other hand, it suggests that the way to find what the something better might be is actually by the kind of refinement which took place in UNIX with wide circulation (among licensees in that case, but presumably among everyone in the T<sub>E</sub>X case).

It strikes me that any form of T<sub>E</sub>X support will have its cost underwritten either by an organization seeking to profit from it or by the Users Group seeking to keep to Knuth's idea of a single uniform T<sub>E</sub>X. My feeling is that the cost, especially to academic users, of the latter is prohibitive (one estimate mentioned at the TUG Steering Committee meeting was \$25,000-50,000/year total distributed among 50-100 institutional members).

My guess is that the costs of making fixes to all releases, i.e. all architectures, could climb above this just because most sites will not have expertise in all systems.

I am afraid that any commercial organization which assumed "official" responsibility for repairing T<sub>E</sub>X would insist on reasonable assurances that no one would compete with them, for example that I would not give away bug fixes the way UNIX sites do. Since no such assurance is possible because the software is in the public domain, it seems that the alternative is to have the entire TUG membership pay for the support by membership fees.

I would propose that as a group TUG provide no services other than the dissemination of information about T<sub>E</sub>X, including the Pascal release. Any bug fixes would be reported but not endorsed by TUG perhaps except at stated intervals when new releases would incorporate them. Production users of T<sub>E</sub>X would be entirely on their own in finding support at the level they need. I am inclined to argue that there should be several distribution sites, one for each architecture and that they should not be sites which have a commercial interest in selling supported T<sub>E</sub>X. They should be reimbursed for their direct expenses by each recipient in the form of a nominal (\$50-200) fee for providing the release tape and documents, and some attempt should be made to ascertain a reasonable level at which TUG will annually reimburse them for less tangible related expenses, e.g. time spent consulting with people having difficulty installing the release, time spent evaluating bug reports, etc. The balance of TUG's budget should be spent for the "switchboard", the newsletter, and the expense surrounding incorporating bug fixes at the stated (infrequent) intervals.

Ordinary TUG membership meetings should be financed largely out of meeting fees which should be appreciably smaller for educational users than for production users.

There will be two classes of institutional memberships: educational users and production users. An educational user is a non-profit educational site which is using T<sub>E</sub>X only for instruction or for the production of publicly accessible research or instructional documents. A production site is a site which is using T<sub>E</sub>X principally for the production of administrative, clerical, or commercial or published documents. A production site is using T<sub>E</sub>X because it expects to save or make money by doing so, whether directly or indirectly. Such sites include not only commercial enterprises, but also the AMS, the publications departments of universities, and the in-house document preparation centers of not-for-profit research organizations.

\* \* \* \* \*

**T<sub>E</sub>X SUPPORT**

Samuel B. Whidden  
American Mathematical Society

In the preceding article, Bob Morris presents his position on TUG's role as it relates to the maintenance and support of T<sub>E</sub>X. As I understand his comments, Bob feels that T<sub>E</sub>X should be left to evolve independently at those computer science departments which have the resources to maintain it. He takes the position that T<sub>E</sub>X is in the public domain for the benefit of the educational user, and that the production user must make his own arrangements for software support of T<sub>E</sub>X.

Bob's comments represent fairly, I think, the opinions expressed by a majority of the Steering Committee at its recent meeting in San Francisco. Most of those present represented the research community, coming either from universities or from the research departments of large corporations.

I believe that if TUG does adopt the position articulated by Bob, it may limit the ability of production users to make use of T<sub>E</sub>X. I stress these two points:

1. If no central maintenance facility exists from which production users can purchase the required software support, potential production users may be unlikely to become users at all—especially those firms which have no need otherwise to acquire the necessary systems programming resources. In those cases, T<sub>E</sub>X's cost may have been raised beyond a reasonable level for other than large firms with existing research programs (or universities, for whom systems programming is among the most available of resources).

2. If undisciplined evolution of T<sub>E</sub>X occurs, the hope of easy communication of machine-readable T<sub>E</sub>X between centers (authors and publishers, for instance) may be dashed. It's unlikely that a common interchange language can be maintained in spite of a proliferation of versions of T<sub>E</sub>X.

It is hard to overemphasize the importance of cost reduction in scientific publishing, nor the importance to the academic community of stable and viable publishing channels. Dick Palais, Chairman of TUG and a Trustee of the American Mathematical Society, calculated some years ago that a total elimination of the tasks of copy-editing, retyping, proofreading, and correcting manuscripts would cut nearly in half the costs of publishing the Society's journals (and one would expect that that statistic could be generalized roughly to other scientific journals). In the last issue of TUGboat,

Ellen Swanson, the Society's Director of Editorial Services, described those costly steps in detail.

The Society embarked on its strong support of T<sub>E</sub>X in the hope that at least some reduction in these tasks ultimately could result from the ability of authors to submit machine-readable, debugged T<sub>E</sub>X-input manuscripts to publishers in the language of T<sub>E</sub>X (and not because other mathematical-composition systems don't exist—they do, and the Society uses one of them).

It seems very unlikely that the standardization necessary to bring about these cost savings can be accomplished without a central coordinating facility. It seems equally unlikely that such a facility can come into being without the active, continuing financial support of the T<sub>E</sub>X user community—TUG—which, at least at present, consists almost exclusively of educational users. It's my personal opinion that such support would have to amount, at least for the time being, to something like \$1,000 per year on the part of each institutional user, to support a TUG budget of \$25,000 to \$40,000 for each of the next two or three years.

These contributions (in the form, perhaps, of contract fees for software support), would be collected by TUG and applied to the various functions needed to keep T<sub>E</sub>X alive, growing, and responsive to changing needs, ideas, technology—all the things that good software stays in touch with. An essential role of the central maintainer, in addition to bug fixing, program distribution, information exchange and telephone consulting, will be to continue to weave into the definitive version of T<sub>E</sub>X the improvements and new features conceived by users—what I'll call here the process of "dynamic standardization". This way, the language grows; if it can't, it probably withers.

The Steering Committee can find volunteer help for some tasks, like publishing the newsletter, but it will almost certainly, if this standardization is to be achieved, have to pay either a competent employee (housed, perhaps, at some willing university) or a "software" firm of some description actually to do the real maintenance work.

Other ways of providing the necessary support were suggested at the Steering Committee meeting. One was for each production user to provide itself with its own systems programming capability in enough depth to support its version of T<sub>E</sub>X. But most small or medium-sized production users (possibly, the majority), even those like the AMS which have some applications-programming capability, won't otherwise need to undertake operating system maintenance on the

level likely to be required for the kind of  $\TeX$  software support which will allow enhancements generated in the user community to be incorporated. To have to acquire such a competence just for  $\TeX$  might price  $\TeX$  beyond their reach; not to have it leaves the user with an increasingly provincial version of  $\TeX$ .

A second possibility suggested for " $\TeX$  Central" was to persuade some user to accept responsibility. Since the very concept of a widely shared language implies the merging of the interests of a diverse group of users, an undertaking to support such a language in any meaningful scope would mean a substantial commitment, almost certainly beyond the resources of, say, the AMS by itself. It is even unlikely that any single university computer science department could commit itself to such an undertaking. The more this alternative is considered, the more it appears that the most reasonable approach to the centralized support and standardization of  $\TeX$  is through the user community as a whole.

A third approach offered was creation of a separate (or TUG sub-) organization, composed only of production users, and for these together to support the central  $\TeX$  software support facility they need. But no active body of production users yet exists, and even if it did, that solution would be likely to provide for standardization of the language among commercial  $\TeX$  users (publishers) while tending to allow dialects to proliferate among educational users, where most authors of journal articles reside.  $\TeX$  as a language of communication between author and publisher would still be unrealizable.

A fourth suggestion was to freeze  $\TeX$  in its present form, and refuse to allow changes to destabilize it. Don Knuth will soon stop making alterations and improvements to  $\TeX$ . When he does, AMS could probably, as the holder of the  $\TeX$  logo copyright, maintain an essentially static "official"  $\TeX$  system, requiring use of only that system for computer-readable manuscript submissions to it. This would limit submissions to those from authors at installations willing to ignore all other versions of  $\TeX$ , no matter what advantages those versions might have acquired, or willing to maintain both our "fixed" version and whatever other evolving versions they chose. In such circumstances, fewer and fewer computer-readable manuscripts would appear (if any ever did), until the Society found itself using  $\TeX$  only for its own internal purposes, much as it now uses other computer typesetting systems. The Society would have gained whatever improvement in typesetting quality  $\TeX$  might represent over earlier

systems, but would have lost, with other scientific publishers, the chance to cut that large portion of its costs representing manuscript re-preparation.

A final suggestion would have a commercial firm take over  $\TeX$  as a software product acting as a vendor, charging a fee for initial distribution of the programs and an additional fee annually for software support. This idea would be good except that, since  $\TeX$  is in the public domain, it's not likely that any firm would offer to involve itself without some sort of endorsement, if not financial guarantee, from TUG. Such an arrangement might work, but only if TUG were willing to put itself on the line to ensure it. Pursuing this idea further might suggest the formation of a small, non-profit organization, under TUG and backed financially by it, to perform  $\TeX$  Central services.

I hope that the members of TUG, when the Steering Committee meets at Stanford in May, will see it in their interest to give real financial support to this effort. If TUG does not make the effort to convert  $\TeX$  to a production system, then it will probably not be converted (except for some specialized classes of users), and potential production users will not be convinced that  $\TeX$  represents a viable choice.  $\TeX$  will remain an educational tool, available in universities, and perhaps in places like AMS where it is used for its competence at certain kinds of typesetting, but it will not become the communications channel which we had hoped for.

It is important that you express your opinions on this subject, no matter what your point of view. Write to TUGboat; your letter will be part of a report to the Steering Committee in May. As many letters as possible will be published in the next issue, which will report on the May meetings.

\* \* \* \* \*

#### UPDATE ON PASCAL METAFONT Scott Kim

Work has just started on the Pascal implementation of METAFONT, which was originally written in SAIL. It is too early to estimate how long the translation will take—stay tuned to TUGboat for news as it develops. Those interested in keeping up with new developments in METAFONT, or knowing more about digital typeface design in general, are invited to correspond with Scott Kim at the Stanford Computer Science Department.

\* \* \* \* \*

## Interface Software

\* \* \* \* \*

**ERRATUM:  
THE FORMAT OF T<sub>E</sub>X'S DVI FILES**

David Fuchs

Editor's note: Several erroneous values were given in a command description which appeared in TUGboat Vol. 1, No. 1, page 18. The full corrected entry is given below.

Command	Description
---------	-------------

129 BOP n&lt;4&gt; p&lt;4&gt;

Beginning of page n, with pointer p to the BOP command of the *previous* page. By "pointer" is meant the relative byte number within the DVI file, where the first byte (the BOP of the first page) is byte number zero. (ex.: If the first page had only a BOP and EOP, the third page's pointer would be 10, because the BOP command takes bytes 0 to 8, the EOP is 9, so the second page's BOP is in byte 10. Get it?). The first page has a -1 for a pointer; the second, a zero. Start the H- and V-coordinates out at 0, as well as the w-, x-, y-, and z-amounts. The stack should be empty, and no characters will be set before a FONT(NUM) command occurs. Remember that n can be < 0, if the page was Roman Numbered. Also the pages need not come in the proper order in the file, depending on who's doing the T<sub>E</sub>Xing.

\* \* \* \* \*

**T<sub>E</sub>X-PASCAL AND PASCAL COMPILERS  
(A STATUS REPORT)**

Ignacio Zabala

Following the original SAIL program, T<sub>E</sub>X-Pascal has suffered some modifications since our previous report of September 1980 (see TUGboat Vol. 1, No. 1, p. 16), but the compiler requirements stated then are still perfectly valid.

The information that has reached us about the latest release of the program indicates three sources of difficulties for some installations:

- The TFX (font information files) that were distributed contained packed information in units of 36 bits.

- The program uses packed records that have to be stored in a single machine word.
- CASE statements contain a default case which is not standard in Pascal.

The first problem is solved in the next release, because T<sub>E</sub>X now uses TFM (T<sub>E</sub>X font metrics) files, which have only 32 bits of information per word.

The second problem, which really affects the amount of memory employed by the program, is not easy to solve without slowing T<sub>E</sub>X down. Essentially, the T<sub>E</sub>X-Pascal program expects that a structure of the type:

```
memoryword = PACKED RECORD CASE 1..4 OF
  1: (pnts: REAL);
  2: (int: INTEGER);
  3: (twohalves: halves2);
  4: (fourbytes: bytes4)
END;
```

where

```
halves2 = PACKED RECORD
  lword: 0..65535;
CASE 1..2 OF
  1: (rhword: 0..65535);
  2: (byte2: 0..255;
     byte3: 0..255)
END;
```

and

```
bytes4 = PACKED RECORD
  byte0: 0..255;
  byte1: 0..255;
CASE 1..2 OF
  1: (rhword: 0..65535);
  2: (byte2: 0..255;
     byte3: 0..255)
END;
```

is stored in a single 32-bit word. It does not make any assumption on the order in which fields are stored: if they are put in the structure and retrieved from it in exactly the same order, the result should be correct.

If the compiler does not pack as expected, it does not necessarily mean that the program will not run correctly, but that it may require an enormous amount of memory (and this may cause the program not to run at all!!).

Some installations that found the third problem (CASE statements) have solved it by hand-editing the program, and it is not yet clear whether it will be fixed once and for all in the sources.

In all, the results are encouraging. Besides Stanford, there are reports of the program running at four other sites:

- On an IBM 370/3033 with Pascal/VS at Stanford CIT (Eagle Berns).
- On a VAX (VMS) at Oregon Software (Barry Smith).
- On an IBM 370/3022 (VM-CMS) with SLAC-Pascal at the University of Pisa (Gianfranco Prini). They printed the DVI files on a Versatec.
- On a Univac 1100/82 at the University of Wisconsin (Ralph Stromquist). Output is to a Compugraphic 8600. (See report, p. 51.)

From the information sent to Stanford, we gather that the Pascal compilers being employed in the installations of T<sub>E</sub>X are:

IBM 370: Pascal-VS, SLAC-Pascal, Pascal-8000

UNIVAC: U. of Wisconsin Pascal, Pascal-8000

PDP-10: Hamburg Pascal

VAX: (See report by Janet Incerpi, p. 49.)

Note: Charles Lawson (Jet Propulsion Lab.—Caltech) has produced two short reports that can help in reprogramming the SYSDEP module of T<sub>E</sub>X-Pascal. (Both are reprinted in this issue, pp. 20 and 32.)

\* \* \* \* \*

## T<sub>E</sub>X FONT METRIC FILES

— or —

What happens when you say "\font A=CMR10"

David Fuchs

When you tell T<sub>E</sub>X that you will be using a particular font, it has to find out information about that font. It gets this information from what are known as .TFM files. For instance, when you say \font A=CMR10 to T<sub>E</sub>X (\A=CMR10 in the old T<sub>E</sub>X lingo), T<sub>E</sub>X looks around for a file called CMR10.TFM, and reads it in. If CMR10.TFM is not to be found, T<sub>E</sub>X will give you the error message "Lookup failed on file CMR10.TFM", and you will be out of luck as far as using CMR10 is concerned.

What does T<sub>E</sub>X want with the TFM files? Generally speaking, a font's TFM file contains information about the height, width and depth of all the characters in the font, plus kerning and ligature information. So, CMR10.TFM might say that the lower-case "d" in CMR10 is 5.55 points wide, 6.94 points high, etc. This is the information that T<sub>E</sub>X uses to make its lowest-level boxes—those around characters. See the T<sub>E</sub>X manual (p. 41) for information about what T<sub>E</sub>X does with these boxes. Note that TFM files do NOT contain any device-dependent description of the font (such as the raster description of the characters at a certain resolution). Remember that the program T<sub>E</sub>X does not deal with

pixels. Only device-drivers that read T<sub>E</sub>X's DVI output files use that sort of information.

Where do .TFM files come from? The best way to get a TFM file is with METAFONT. Font designers should include the METAFONT instructions that specify the width, height, etc. of each character they design. The METAFONT manual contains details and examples of how to do this—see the index entries for charwd, charht, chardp, etc. If this is done, then when METAFONT is run on CMR10, it produces CMR10.TFM. (Depending on what "mode" it is run in, it also makes CMR10.FNT, CMR10.ANT, CMR10.VNT, or CMR10.OC. These are all different formats for files containing the raster description of the font. Drivers for various devices require one or another of these files.)

Whatever happened to the TFX format that the T<sub>E</sub>X and METAFONT manuals actually refer to? Is this just a misprint for TFM? No—TFM files take the place of TFX files. The differences are conceptually small; they both contain more or less the same information. The main reason for changing T<sub>E</sub>X and METAFONT from using/making TFX files to TFM files is that TFX files were based on 36-bit words. This proved to be a real problem for people running Pascal T<sub>E</sub>X, especially on 32-bit machines. The format of TFM files assumes 8-bit bytes, packed four to a 32- or 36-bit word. They are readily adapted for use on 16-bit machines, too. While the format was being changed, a few new bits of information were added, too.

What if I have fonts that I want T<sub>E</sub>X to know about that were not made with METAFONT? Don't despair—we have two programs, TFTOPL and PLTOTF, that convert TFM files to readable, editable format, and back again. For instance, if we run TFTOPL on CMR10.TFM, it makes CMR10.PL, an excerpt of which follows ("R" means a floating-point number is coming up (all dimensions given in this form are in terms of the DESIGNSIZE); "C" means that a character is next; "O" means an octal value for a character that isn't an ASCII printing character is next):

```
(FAMILY CMR)
(DESIGNSIZE R 10.000000)
(CODINGScheme TEX TEXT)
(TEXINFO
  (SPACE R .3333330)
  (STRETCH R .1666670)
  (SHRINK R .1111107)
  (XHEIGHT R .4444447)
  (QUAD R 1.000000)
  (EXTRASPACE R .1111107)
)
```

## (LIGTABLE

```
(LABEL C f)
(LIG C 1 0 174)
(LIG C f 0 173)
(LIG C 1 0 175)
(LIG C t 0 44)
(STOP)
(LABEL C o)
(KRN C o R .0277777)
(KRN C x R -.0277777)
(STOP)
)
```

## (CHARACTER C d

```
(CHARWD R .5555553)
(CHARHT R .6944447)
(CHARDP R .0000000)
)
```

Which says: This font is in the CMR family, its size is 10.0 points, and it's a regular TeX font for text. When TeX uses this font, it should know that the glue between words should be 3.33pt plus 1.66pt minus 1.11pt. TEX will also know that the x-height of CMR10 is 4.44 pt, a "quad" of space is 10 pt, and the extra amount of space at the end of a sentence is 1.11 pt. (see the METAFONT (Appendix F) and TeX (everywhere) manuals for more on these parameters). Also, TeX should recognize that whenever "f" is followed by "i", "r", "l", or "t", that it's time for a ligature: an occurrence of "fi" should be replaced by the character in octal position 174 in CMR10 ("fi"; see the Appendix of the TeX manual that shows font tables to verify this), etc. Whenever "o" is next to another "o", they should be moved apart by .277 pt more than they would otherwise be (based on their widths), but an "o" should be moved .277 pt closer to a following "x". Finally, the character "d" has width 5.55 pt, height 6.94 pt, and depth 0 pt. The full CMR10.PL file has lots more in its LIGTABLE and many more CHARACTER descriptions, of course.

If we changed the line

```
(CHARWD R .5555553)
```

to

```
(CHARWD R .700)
```

and ran PLTOTF, making a new CMR10.TFM, TeX would now think that "d" had width 7 pt in CMR10.

So, if you have your own fonts that you'd like TeX to know about, just make PL files for them, and then run PLTOTF to make TFM files. With luck, your fonts will be in some computer readable form such that the PL files can be made with a fairly simple program. Note that it is perfectly legal to have a TFM (or PL) file that specifies no kerns or ligatures.

In fact, TFM files may contain a fair amount more information about a font than just the heights, widths, depths, kerns and ligatures. Most of these extra parameters only come up in the special math fonts. The METAFONT (Appendix F again) and TeX manuals talk about these parameters in more detail. Below is a complete but scary description of all the bits in TFM files. This description was actually excerpted from a comment in the SAIL-language version of TeX.

\* \* \* \* \*

This definition of TFM files is due to Lyle Ramshaw.

Each font used by TeX has an associated font information file. The name of this file is obtained by appending the extension code ".TFM" to the font file name. For example, the TeX font metrics for the font CMR10 appear on the file CMR10.TFM. These .TFM files are written with 32 bits in each word, to facilitate their transportability. When they sit in the file systems of 36-bit machines, these 32 data bits will be left-justified in the 36-bit word, leaving the rightmost four bits zero.

The first 6 words of the .TFM file contain twelve 16-bit integers that give the lengths of the various portions of the file, packed two to a 32-bit word. These twelve integers are, in order:

```
lf = length of entire file in words,
lh = length of header data,
bc = first character code in font,
ec = last character code in font,
nw = number of words in width table,
nh = number of words in height table,
nd = number of words in depth table,
ni = number of words in italic correction table,
nl = number of words of lig/kern program,
nk = number of words in kern table,
ne = number of words in extensible character
    table,
np = number of font parameters.
```

In .TFM format, the subfields of a word are always allocated in left-to-right (BigEndian) order. Thus, the first two integers in this list, lf and lh, are packed into the first word of the .TFM file with lf on the left and lh on the right.

These lengths are not all independent: they must obey the relation

$$lf = 8 + lh + (ec - bc + 1) + nw + nh + nd + ni + nk + nl + ne + np.$$

The rest of the .TFM file is a sequence of ten data arrays as specified below:

```

HEADER= ARRAY[0:lh-1] of Stuff
FINFO= ARRAY[bc:ec] of FInfoEntry
WIDTH= ARRAY[0:nw-1] of FIX
HEIGHT= ARRAY[0:nh-1] of FIX
DEPTH= ARRAY[0:nd-1] of FIX
CHARIC= ARRAY[0:n1-1] of FIX
LIG/KERN= ARRAY[0:n1-1] of LigKernStep
KERN= ARRAY[0:nk-1] of FIX
EXT= ARRAY[0:ne-1] of ExtRecipe
PARAMS= ARRAY[1:np] of FIX.

```

A FIX is a one-word representation of a real number in a fixed-point fashion. FIXes are used in .TFM format to enhance transportability. A FIX is a signed quantity, with the two's complement of the entire FIX used to represent negation. Of the 32 bits in the word, 12 are to the left and 20 are to the right of the binary point. This means that a FIX has 1 bit of sign, 11 bits of integer, and 20 bits of fraction. Note that this limits the size of real numbers that a FIX can represent: the largest FIX is roughly 2000.

The first data array is a block of header information, general information about the font. Currently, the header contains 18 words, allocated as described below. In the future, new fields might be added at the end of the header block.

```

HEADER=
[
Checksum: 1 word
DesignSize: FIX (1 word)
CharacterCodingScheme: 10 words
ParcFontIdentifier: 5 words
Random word (=Header[17]
is broken up as follows:)=
[
SevenBitSafe: 1 bit
unusedspace: 23 bits
ParcFaceByte: 8 bits
]
]

```

The Checksum field is used to hold a unique identifier of some sort that describes this version of the font. This unique ID is put by METAFONT into both the rasters and metrics. TeX finds it in the metrics, and stores it in the .DVI file. Thus, a spooler can check the unique ID in the .DVI with the unique ID in its rasters, to provide a guarantee that TeX was working with metric data for the current rasters. METAFONT computes this checksum from the metric information in the .TFM file.

The DesignSize of the font is the size that the font was intended to look good at, or, to put it another way, the nominal size of the font when it is printed at a magnification of 1.0. For unusual fonts

such as CMDUNH and CMATHX, the DesignSize is more-or-less arbitrary. The DesignSize is stored as a FIX with the units "points".

The CharacterCodingScheme field is supposed to specify what the character code to symbol translation scheme is in this font. The coding scheme is stored in 10 words=40 bytes of the .TFM file, as a string. The first byte gives the length of the string, the next  $n$  bytes are the characters, and the last  $(39 - n)$  bytes are zeros (where "first" and "next" imply working from left-to-right). Some common coding scheme names are:

#### CharacterCodingSchemes:

```

TEX TEXT
TEX TYPEWRITER TEXT
TEX MATHIT
TEX MATHSY
TEX MATHSX
UNSPECIFIED — default, means no information
GRAPHIC — special purpose code, non-
alphabetic
ALPHABETIC — means alphabet agrees with
ASCII at least
ASCII — means exactly ASCII
PARC TEXT — Times Roman and Helvetica,
for example

SUAI
CMU
MIT

```

Fonts are universally referred to by their file names: for example "CMR10" for Computer Modern Roman 10 point, "CMTT" for Computer Modern TypeWriter Type 10 point, etc. The TeX user specifies the font by giving this string name, the TeX output module finds the metric file by using this name with the extension .TFM, and the various printers store the rasters as files with this name and some other extension.

TeX can only handle character codes that are seven bits in length. But the .TFM format always allows a full eight bits for a character code, so the high-order bit of all characters specified must be zero.

The HEADER data is followed by the FINFO table, which is an array of FInfoEntry's. This array is indexed from bc to ec, and hence contains  $(ec - bc + 1)$  entries. Each FInfoEntry is one word in length. An FInfoEntry is a compacted structure with the following format (fields allocated from left to right once again):



**FInfoEntry:**

```
[
  WidthIndex: 8 bits
  HeightIndex: 4 bits
  DepthIndex: 4 bits
  CharIcIndex: 6 bits
  TagField: 2 bits
  Remainder: 8 bits
]
```

The fields in the **FInfoEntry** do not give the character width, height, etc. directly, they are indices into secondary tables. Thus, up to 256 different widths may appear among the 256 characters of a single font, and up to 16 different heights, 16 different depths, and 64 different italic corrections. The actual widths, heights, depths and italic corrections are stored in the **.TFM** file as arrays of **FIXes** with "em" as their units. **TeX** reads in these **FIXes**, converts them to floating point form, scales them by multiplying by the desired font size (in points), and stores them into internal character metric arrays.

The **CharIc** field is used both for the italic correction of ordinary characters and for mathop kerns. In particular, for mathops such as summation and integral signs, the **CharIc** field points to a "kern" which, if nonzero, means that limits are normally set to the right and the lower limit is shifted left by this kern value. If the kern is 0, limits in display style will be centered above and below the operator. (To change between centering and attaching at the right, one writes "\limitswitch" after the operator.)

A note on non-existent characters: all character codes outside of the range [bc, ec] represent characters that do not exist in the font. Any codes in the range [bc, ec] that represent non-existent characters will have their **FInfoEntry**s identically equal to 0. The **WIDTH**, **HEIGHT**, **DEPTH**, and **CharIc** arrays will each be guaranteed to have a **FIX** of 0.0 in their 0'th position. Thus, failing to notice that a character is non-existent won't lead a program to use irrelevant metric data for that character code. Furthermore, any characters that really do exist in the font will be guaranteed to have a **WidthIndex** that is nonzero. Thus, a character is non-existent iff its **WidthIndex** is zero, and also iff its entire **FInfoEntry** is zero. If there are any actual characters in the font whose width just happens to be precisely zero, the **WIDTH** array will contain two zero **FIXes**: one at index 0, which is used for all of the non-existent characters, and one somewhere else.

The remaining portion of the **FInfoEntry** is used for several different purposes, depending upon the value of the tag field. The **TagField** portion of the **FInfoEntry** has one of four values:

```
tag=0 => this is a vanilla character, Remainder is
          unused.
tag=1 => character has a ligature-kerning program:
          the Remainder field is the index in the
          LIG/KERN array of the first step of the pro-
          gram.
tag=2 => character is part of a chain of charac-
          ters of ascending sizes ("charlist"): the
          Remainder field gives the character code
          of the next larger character in the chain.
tag=3 => character code represents an extensible
          character, one that is built up out of
          smaller pieces and can be made arbitrarily
          large: the Remainder field is an index into
          the EXT array. The ExtRecipe at that
          position in the EXT array describes what
          the pieces are.
```

(The **taglist** and **tagvar** options are usually used only in math extension fonts.)

The **LIG/KERN** array is a program in a simple programming language that gives instructions about what to do for special letter pairs. Each step in this program occupies one word:

**LigKernStep:**

```
[
  StopBit: 1 bit
    # means this is a final program step
  unusedspace: 7 bits
  NextChar: 8 bits
    # if this is the next character, then...
  TagBit: 1 bit
  unusedspace: 7 bits
  Remainder: 8 bits
]
```

If the **TagBit** is 0, this step in the program describes a ligature. In that case, the **Remainder** consists of the character code of the ligature that should be substituted for the current character pair. If the **TagBit** is 1, this step describes a kern, and the **Remainder** field is an index into the **KERN** array. The **KERN** array is simply an array of **FIXes**, pure numbers that should be scaled to give distances in the same way as the elements of the **WIDTH**, **HEIGHT**, **DEPTH**, and **CharIc** arrays.

An **ExtRecipe** is a one-word quantity that should be viewed as four bytes (allocated left-to-right, of course):

**ExtRecipe:**

```
[
  top: byte
  mid: byte
  bot: byte
  ext: byte
]
```

The height and width fields in the `FInfoEntry` of the extensible character give the metrics of the component, not of the built-up symbol itself, since the built-up symbol will have variable size. If top, middle, or bottom portions are zero, the extension component runs all the way through that portion of the symbol, otherwise it directly abuts these portions. The built-up symbol is formed by including an integral number of extension components. If there is a middle, the same number of extension components will appear above and below. For example, a left brace has all four components specified, while a double `||` (the cardinality or norm symbol) has only an extension part. The floor and ceiling brackets are like regular brackets, but without top or bottom, respectively. The width of the extension component is assumed to be the width of the entire built-up symbol. If any byte is 0, it indicates that the corresponding piece of the extensible character does not exist. Otherwise, the contents of the byte is the character code of the piece: top, middle, bottom, or extender respectively.

The rest of the `.TFM` file is the `PARAMS` array, a table of font parameters that are used by `TEX`, stored as `FIXes`. All of these parameters are distances except for the first one, "slant": hence all except for "slant" should be scaled by the font size by `TEX` when being read in from the `.TFM` file. Since `slant` is a pure number, it should not be scaled. [The following table of parameters is printed in clearer form on pages 98–100 of the `METAFONT` manual.]

**slant** the amount of italic slant (e.g. `slant=.25` means that when going up one unit, go .25 units to the right—this is used in placing accents over characters)

**space** a real number that says how wide blank spaces are (Note that `TEX` doesn't use character number '40 for spaces, that character can be non-blank in the font)

**spacestretch** the stretch component of the glue for spacing

**spaceshrink** the shrink component of the glue for spacing

**xheight** the height of lowercase "x" (default positioning for accents)

**quad** the width of one "em"

**extraspace** the amount added to `space` after periods (and in general when the `spacefactor` is greater than 2)

Mathematics fonts used as `\mathsy` and `\mathex` contain important additional parameter information. In a `\mathsy` font, the extra parameters start right after "quad", that is, there is no "extraspace" parameter. The `\mathsy` parameters are

**mathspace** if nonzero, the amount of space that will be used for all nonzero space in math formulas (for fixed-width output)

**num1, num2, num3** amount to raise baseline of numerators in display or nondisplay or nondisplay-atop styles, respectively

**denom1, denom2** amount to lower baseline of denominators

**sup1, sup2, sup3** amount to raise baseline of superscripts if  
1) display style  
2) nondisplay nonvariant style  
3) variant style

**sub1, sub2** amount to lower baseline of subscripts if superscript is  
1) absent  
2) present

**supdrop, subdrop** amount below top or bottom of large box to place baseline if the box has a superscript or subscript in this size

**delim1, delim2** size of `\comb` delimiters in  
1) display  
2) nondisplay style

**axisheight** height of fraction lines above the baseline (this is midway between the two bars of = sign)

A `\mathex` font includes the first seven standard parameters (including `extraspace`), and then has six parameters used to govern formula setting:

**defaultrulethickness** the thickness of `\over` and `\overline` bars

**bigopspacing(1), (2)** the minimum glue space above and below a large displayed operator, respectively

**bigopspacing(3), (4)** the minimum distance between a limit's baseline and a large displayed operator, when the limit is above, below

**bigopspacing(5)** the extra glue placed above and below displayed limits

## TeX Support Programs

by

Phil Sherrod and Alan Wright  
Vanderbilt University

The process of installing TeX on a computer system involves more work than just getting the TeX program itself running. It is also necessary to have support programs to convert the device *independent* output produced by TeX (the "DVI" files) into a device *dependent* form suitable for driving one or more output devices.

A DECSys-10 computer is used at Vanderbilt which is very similar to the DECSys-20 used at Stanford. This enabled us to get the SAIL version of TeX running with only a few days of effort. During the year that we have been using TeX we have found it to be remarkably reliable and bug-free and have had to devote very little time to its maintenance. Most of our effort has been directed toward improving the efficiency and convenience of the support programs used to drive the Versatec printer.

The Versatec printer is not connected directly to the DEC-10 but rather is connected by a parallel port to a Monolithic Systems Corp. Z80 micro-computer system which consists of a Z80 processor and 64Kb of memory. This system serves as an intelligent controller for the Versatec and unloads a significant amount of processing from the DEC-10. The Monolithic micro is connected to the DEC-10 through a 9600 baud RS232 serial line.

The support programs provided by Stanford for driving a Versatec consisted of two DEC-10 programs written in SAIL, DVIVER and VERSER, and an assembly language program for the Z80 called PAINT. We obtained a cross assembler for the Z80 from Stanford and wrote our own program to communicate with the ROM monitor in the Z80 and downline load program images.

The DVIVER and VERSER programs were moderately large (about 60K 36-bit words) and fairly slow, consuming a total of about 7 seconds of CPU time per page generated (TeX only uses about 1.5 seconds of CPU time in formatting each page).

The DVIVER program reads the DVI files produced by TeX, breaks tall characters into "parts" which will fit in the raster-scan buffer area in the Z80; sorts the parts on each page into top-to-bottom order and produces an intermediate

work file that is read by **VERSER**. **VERSER** reads this work file and produces a file of commands and data to be transmitted to the Z80. The actual process of generating the raster scan pattern for each character and part takes place in the Z80. However, since the Z80 has no local storage, **VERSER** must downline load the raster pattern definition for each character as part of the command and data file it produces for a document. Once a raster pattern has been defined for a particular character the invocation of it requires only a simple and short command.

The total number of character pattern definitions that can be simultaneously held in the Z80 is constrained by the available memory space in the Z80. In the original implementation of the Z80 **PAINT** program, each character (or part) definition occupied a fixed size of 256 bytes in Z80 memory. The **VERSER** program kept track of which character definitions were currently defined in Z80 memory and used a round-robin technique to replace old character definitions with new ones if more characters were used than could be simultaneously accommodated by the Z80. This meant that any number of characters could be used in a document but a character definition might have to be reloaded several times.

Our work at Vanderbilt on the **TeX** support programs has been directed at improving their efficiency and convenience. The **DVIVER** and **VERSER** programs were combined into a single program called **VERTEX** which was written in **BLISS** for efficiency. **VERTEX** performs the same basic functions as **DVIVER** and **VERSER** but is more efficient because it does not have to generate the intermediate work file and can avoid having to reaccess font information that was used by both **DVIVER** and **VERSER**. Its memory management is also more efficient and it uses a least-recently-used technique to keep in memory information about fonts. The **VERTEX** program occupies 33K words of memory and uses about 5 seconds of CPU time per page generated.

The Z80 **PAINT** program was also rewritten. The memory space occupied by instruction code was reduced to 30 percent of its original size allowing more space to be used for character pattern definitions. The space allocated for each character pattern definition was reduced from 256 to 128 bytes. This resulted in more characters having to be broken into parts but substantially increased the total number of characters that could simultaneously be defined in Z80 memory. A least-recently-used (rather than round-robin) technique is used to select character definitions to be replaced when more characters are needed than can be defined simultaneously in Z80 memory. The command protocol used to communicate between the DEC-10 and the Z80 was also improved. A reinitialization command sequence was added that allows the Z80 to be restarted under DEC-10 control; status information about paper and toner supplies are sent from the Z80 to the

## DEC-10.

The last part of our project was to integrate the TeX support programs into the spooling system on the DEC-10. We did this by adapting VERTEX to fit into the GALAXY spooling system which is part of the operating system on the DEC-10. A time-sharing user can now issue a "VPRINT dvi-file" command to queue a file for printing on the Versatec. The conversion of the DVI file to Z80 commands is performed by the spooler; the Z80 commands are transmitted directly to the micro-processor without creating any intermediate files.

As part of our rewrite of the PAINT program, we added commands to do general vector drawing. A DEC-10 program has been written to display on the Versatec plot files created in a format compatible with our Zeta pen plotter. We intend to modify TeX to add a command which will specify the name of a plot file and the amount of space to reserve for it. VERTEX will then merge the plot commands with the document as it is being processed for printing on the Versatec.

Brief Functional Characterizations of the  
Procedures in the T<sub>E</sub>X/PASCAL Compilation Unit, SYSDEP

by

C. L. Lawson  
I. Zabala  
M. Díaz

Stanford University, January 27, 1981\*

\*This document is an enlarged and revised edition of a paper with the same name, published by C. L. Lawson; Computing Memorandum No. 166, September 10, 1980, Jet Propulsion Laboratory, California Institute of Technology.

# Contents

	Page
Introduction . . . . .	1
1 Handling Printable Characters and Strings . . . . .	1
APPNDREAL . . . . .	1
APPNDSTRING . . . . .	2
INITSTRINGS . . . . .	2
PRINT, PRINTINT, PRINTLN, PRINTOCTAL, and PRINTREAL . . . . .	2
PRODUCESTRING . . . . .	2
2 Handling External File Names . . . . .	2
INITFILENAME . . . . .	2
APPENDTONAME . . . . .	2
PRINTFILENAME . . . . .	3
PRODUCESTRING . . . . .	3
SCANFILENAME . . . . .	3
3 Operations on Input Files ICHANx . . . . .	3
CHANPTR . . . . .	4
EOFCHAN . . . . .	4
GETCHAN . . . . .	4
GETFIRSTLINE . . . . .	4
INLN . . . . .	4
RELEASE . . . . .	4
RSETFILE . . . . .	4
4 Operations on Output Files OCHANx . . . . .	4
CLOSE . . . . .	4
RWRITEFILE . . . . .	5
SENDCH . . . . .	5
SENDLN . . . . .	5
SENDSTARTED . . . . .	5
5 Terminal I/O and Output to the Error File . . . . .	5
FORCEBUFFEROUT . . . . .	5
INCHTER . . . . .	5
INITSYSDEF . . . . .	5
INLNTER . . . . .	5
OUTCHERR . . . . .	6
OUTCHTER . . . . .	6
OUTLNERR . . . . .	6
PRINT . . . . .	6
PRINTFILENAME . . . . .	6

PRINTINT	6
PRINTLN	6
PRINTOCTAL	6
PRINTREAL	6
TRACELINE	6
<b>6 Read Font Information</b>	<b>7</b>
READFONTINFO	7
<b>7 Outputting Initialised Tables from TEXPRES</b>	<b>7</b>
SETTABLESIZES	7
WRITEDELIMTB	7
WRITEEQTB	7
WRITEFMEM	7
WRITEHYPHENTB	7
WRITEPAGETB	8
WRITESECONDMEM	8
<b>8 Reading Initialised Tables into TeX</b>	<b>8</b>
GETTABLESIZES	8
INITDELIMTB	8
INITEQTB	8
INITFMEM	8
INITHYPHENTB	8
INITPAGETB	8
INITSECONDMEM	8
<b>9 Handling the Device-independent Output File, DVI</b>	<b>9</b>
CLOSEOUT	9
DECLAREOFIL	9
DVI	9
INTOUT	9



Brief Functional Characterizations of the  
Procedures in the TeX/PASCAL Compilation Unit, SYSDEP

## Introduction

In working on a Univac computer system with the PASCAL code for TeX, TEXPRES, and SYSDEP, I have felt a need for some types of documentation other than the documentation provided by the Stanford group. I have had telephone discussions with members of the Stanford group on this topic. The purpose of this memo is to provide an specific example of one type of document that I feel would be a very useful auxiliary document for anyone undertaking to install TeX/PASCAL on a different computer system.

I have used the following guidelines in composing this document.

- A. Group the procedures into functionally related classes.
- B. For each group describe the general functional area covered by the group, and any data structures, limited ranges of parameter values, etc., common to the whole group.
- C. Within each group list the procedure names in alphabetic order with a brief functional characterization of each procedure. Format this on the page so the procedure names stand out prominently from the associated text.
- D. Use only terminology that is well-defined in the PASCAL model of computer programming. Any other terminology must be defined in terms of PASCAL or English language primitives. Examples of apparently DEC-10 related terms to be eschewed include "WAITS system", "file extension", "file directory", "directory name", and "system directories".
- E. Include a table of contents so the reader can see the major groupings of procedures at a glance.<sup>1</sup>

## §1 Handling Printable Characters and Strings

The array STRINGPOOL holds strings of printable characters of two types. There is a printable string associated with each ASCII ordinal in the range [0..127], and there is also a miscellaneous set of printable strings for use as error messages, etc. Pointers into the array STRINGPOOL are held in the array STRNG.

### APPNDREAL

Appends a real number to a string (both things are arguments of the function). The rules for the string and for the returned value are the same as those in APPNDSTRING (below). Before appending, the real number is converted into a string of five characters that has a decimal point in the fourth position. Since it will always be positive, there's no need for a sign. For example, the string obtained from 3/2 is precisely "<space><space>1.5".

---

<sup>1</sup>See also, by the same author, *Detailed Specification of Procedures in the TeX/PASCAL Compilation Unit SYSDEP*, J. P. L. Section 366, Memo No. 487.

**APPNDSTRING**

This procedure appends string one string to another, fetching the former from either `FILENAME` or from `STRINGPOOL`.

In any case, the function returns an identifier for the string as required by `PRODUCESTRING`.

**INITSTRINGS**

Reads printable strings for the ASCII ordinals [0..127] from the external file "ASCII TBL", and additional printable strings from the external file "STRINITBL". These are all stored into the array `STRINGPOOL`, indexed from the array `STRNG`.

Reading is done using `ICHAN1` which is reset for each of these two external files.

**PRINT, PRINTINT, PRINTLN, PRINTOCTAL, and PRINTREAL**

These procedures build print images from the strings in `STRINGPOOL` and their arguments, and output them to the terminal as well as to the error file. See Section 5 for the specific function of each of these five procedures.

**PRODUCESTRING**

Fetches a string from either `STRINGPOOL` or `FILENAME`, and returns it to the calling procedure.

**§2 Handling External File Names**

This set of procedures keeps track of external file names and their associations with internal file names. These procedures are very strongly oriented toward a DEC-10 system. It appears that some of this functionality would be handled quite differently on other systems.

External file names are held in the array `FILENAME`. Indexes  $f$  in the `FILENAME` array are usually passed as `FILNAM( $f$ )` to procedures dealing with strings, to help them find out whether the argument should be located in the `FILENAME` array or in `STRINGPOOL`.

**INITFILENAME**

Initializes the data structures that will be employed to parse an external file name. It is called once for each external file name. Its effect depends on whether the given file is an input file, an output file, or one of the font information files.

Refer to the description of function `SCANFILENAME` below.

**APPENDTONAME**

Refer to the description of function `SCANFILENAME` below.

**PRINTFILENAME**

Outputs a file name from the array **FILENAME** to the terminal and the error file.

**PRODUCESTRING**

Fetches a string from either **FILENAME** or **STRINGPOOL**, and returns it to the calling procedure.

**SCANFILENAME**

The function **SCANFILENAME** scans the input for the name of an external file and tries to open it. Its argument says whether it is a font file or send stream for which space has been reserved in the data structures or an input file that has not been allocated yet. This function belongs in the main **TEX** module, but because file names are very system dependent, the parsing is done in the system dependent module as follows:

First **INITFILENAME** is called with the same argument that **SCANFILENAME** received from its caller. **INITFILENAME** should then take care of any initialization needed for the parsing of the file name. Using its argument, it can decide whether the parsed name will refer to a font information file, to a "send" stream or to an ordinary input file. The negative of the argument is returned if there are too many open files (this can only happen with input files).

Next **APPENDTONAME** is called once for each token in the input. Using the data structures set up by **INITFILENAME**, **APPENDTONAME** decides everytime whether the part of the name received so far is syntactically correct for such a file name (otherwise it returns "malformedname") and if so, whether it has already received a complete file name (otherwise it returns "needmore"). In case the whole file name was received, **APPENDTONAME** tries to open that file. If it succeeds, it will return done, otherwise it returns "failure". In any case, **APPENDTONAME** saves the scanned string in a place from where it can be retrieved for use in error messages.

In the Stanford version of **SYSDEP**, some defaults are set for certain portions of certain file names. note **APPENDTONAME** accepts everything until it finds a token that could never be part of a file name. If it is a delimiter (space, period, etc.) it considers that the whole file name was received and tries to open it. Otherwise it returns "malformedname". Upon an unsuccessful return from **APPENDTONAME**, **SCANFILENAME** returns a value that allows its calling routine to recognize the failure and locate the stored name if it has to be printed in some error message.

**§3 Operations on Input Files ICHANx**

These procedures do various operations on the six input files **ICHAN1** through **ICHAN6**. The file is selected by an integer parameter in the range [1..6].

External file names associated with these internal file names are stored in the array **FILENAME** with index values [**MAXFNT**+1+10+1..**MAXFNT**+1+10+6]. This is the last part of the **FILENAME** array, and operates in a LIFO manner. The name of the last \input command is stored in the first free entry in that range. When the end-of-file is reached, that entry is liberated.

**NOTE:** **SYSDEP** uses **ICHANx** with **x** in the range [1..6] because of the existence of a compiler imposed limit on the number of files that a **PASCAL** program can keep open. This is clearly a system dependent parameter. Six input files may not be enough for some complicated **TEX** sources.

**CHANPTR**

Returns the ordinal value of the current input character in `ICHANx`; namely, it executes `CHANPTR:=ORD(ICHANx↑)`.

**EOFCHAN**

Returns the value of `EOF(ICHANx)`.

**GETCHAN**

Executes `GET(ICHANx)`.

**GETFIRSTLINE**

Skips system header stuff, if any, at the beginning of a file to get a position for reading first meaningful information. This is highly system dependent.

**INLN**

Reads one line from `ICHANx`.

**RELEASE**

Releases the input file `ICHANx` by executing `RESET(ICHANx)` followed by `FILPTR:=FILPTR-1`. The latter statement frees the top entry in `FILENAME`, the top of the stack of input files.

**RSETFILE**

Opens `ICHANx` for input using `RESET` with a nonstandard parameter list.

This procedure is completely analogous to the PDP-10 PASCAL reset procedure. Its arguments `FNAME`, `FDIRECTORY`, and `FDEVICE` are required by it precisely in the given form.

**§4 Operations on Output Files OCHANx**

These procedures do various operations on the ten output files `OCHAN0` through `OCHAN9`. The file is selected by an integer parameter in the range `[0..9]`.

External file names associated with these internal file names are stored in the array `FILENAME` with index values `[MAXFONT..MAXFNT+11]`. (Entry `"MAXFNT+1"` is reserved for temporary storage of font names, when the font itself was preloaded by `TEXPRE`)

**CLOSE**

Executes `RESET(OCHANx)` to close a file.

**RWRITEFILE**

Opens file 0CHANx for output using REWRITE with a nonstandard parameter list.

This procedure is completely analogous to the PDP-10 PASCAL rewrite procedure. Its arguments FNAME, FDIRECTORY, and FDEVICE are required by it precisely in the given form.

**SENDCH**

Outputs one character to 0CHANx.

**SENDLN**

Outputs a carriage return and line feed to 0CHANx.

**SENDSTARTED**

Executes SENDSTARTED:=EOF(0CHANx). The result is true only if this output file has been started but not opened.

**§5 Terminal I/O and Output to the Error File**

These procedures handle I/O from and to the terminal and output to the error file. The internal file names used are TERIN, TEROUT, and ERRFIL. The external names for both input and output to the terminal is "FOOBARTTY" and thus TERIN and TEROUT must each be reopened every time there is a switch between input and output or vice versa. The external name of ERRFIL is "ERRORSTEM".

**FORCEBUFFEROUT**

Completes output to the terminal before doing input from the terminal. Uses the nonstandard Pascal function BREAK(TEROUT).

**INCHTER**

Executes BREAK(TEROUT), opens TERIN for input, inputs the ordinal value of one character from TERIN, and skips to a carriage return.

**INITSYSDEP**

Opens TEROUT and ERRFIL and does other initializations.

**INLNTER**

Reads one line from TERIN.

**OUTCHERR**

Outputs one character to ERRFIL.

**OUTCHTER**

Outputs one character to TEROUT.

**OUTLNERR**

Outputs a carriage return and line feed to ERRFIL.

**PRINT**

Outputs a string from STRINGPOOL to TEROUT and ERRFIL.

**PRINTFILENAME**

Outputs a file name from the array FILENAME to TEROUT and ERRFIL.

**PRINTINT**

Outputs a print image of an integer to EROUT and ERRFIL.

**PRINTLN**

Outputs a carriage return and line feed and a string from STRINGPOOL to TEROUT and ERRFIL.

**PRINTOCTAL**

Outputs the octal print image of an integer to TEROUT and ERRFIL.

**PRINTREAL**

Output a print image of REAL number to TEROUT and ERRFIL.

**TRACELINE**

Displays a line to the terminal and the error file. Awaits a signal from the terminal. Anything except a carriage return causes this procedure to read a new line from the terminal replacing the displayed line.

## §6 Read Font Information

Each font used by  $\TeX$  has an associated font information file –called TFM, for  $\TeX$  Font Metrics– that must be read into the arrays FONTINFO, FMEM (updating also the pointer FMEMPTR), WDBASE, HTBASE, DPBASE, ICBASE, LGBASE, KRBASE, EXTBASE, and PARBASE, for internal use by the  $\TeX$  program.

In DEC machines, the name of this file is obtained by appending the extension code “TFM” to the font file name. For example, the  $\TeX$  font metrics for the font CMR10 appear on the file CMR10.TFM.

### READFONTINFO

Takes the font information from a file and puts it in the internal character metric arrays employed by  $\TeX$ . The contents of these arrays is described in *TEX.DOC*. Notice that, sometimes, FONTINFO is set to integer 0, and that FONTFIL.FOURBYTES is assigned directly to FONTINFO entries. This works correctly for the PDP-10 pascal compiler where fields are packed left-to-right in each word (thus, unused bits are always on the right).

## §7 Outputting Initialized Tables from TEXPRE

These procedures output various tables from TEXPRE for later input to  $\TeX$  to initialize  $\TeX$ . The file being written has the internal name TBLFIL and the external name “TEXINITBL”.

Thus,  $\TeX$  does not have to go through all the initialization; it is enough to read the tables from that file.

### SETTABLESIZES

Opens file TBLFIL and outputs twelve numbers giving sizes of tables to follow.

### WRITEDELIMTB

Outputs the delimiters table.

### WRITEEQTB

Outputs the equivalentents and related tables.

### WRITEFMEM

Outputs the font memory.

### WRITEHYPHENTB

Outputs hyphenation tables.

**WRITEPAGETB**

Writes-out original contents of the memory table. This table contains such parameters as `\hsize`, `\vsize`, `\parindent`, `\topbaseline`, `\varunit`, etc.

**WRITESECONDMEM**

Outputs the latter part of the array MEM. It is necessary to initialize this part because it may contain names (character strings) of control sequences that were defined in the files preloaded by TEXPRE.

**§8 Reading Initialized Tables into TeX**

These procedures read various tables into TeX for initialization. The file being read has the internal name TBLFIL and the external name "TEXINITBL".

**GETTABLESIZES**

Opens file TBLFIL for input and reads twelve integers giving sizes of the tables which follow on this file.

**INITDELIMTB**

Inputs the delimiters table.

**INITEQTB**

Inputs the equivalents and related tables.

**INITFMEM**

Inputs the font memory.

**INITHYPHENTB**

Inputs the hyphenation tables.

**INITPAGETB**

Inputs the page memory. Cf. WRITEPAGETB.

**INITSECONDMEM**

Inputs the latter part of the array MEM. Cf. WRITESECONDMEM.



## §9 Handling the Device-independent Output File, DVI

This file contains the main output from  $\TeX$ . Its internal name is `OUTFIL`. Its default external name is `"TEXTOUTDVI"`.

### CLOSEOUT

Called by  $\TeX$  just before stopping to write postamble information on `OUTFIL`. This information is partially system dependent, because it contains font file names.

### DECLAREOFIL

Initializes the output on the file indicated by its argument; that is, sets up correspondence between the internal file name `OUTFIL` and the external name.

This procedure is called when the name of the output file is first known. It then opens file `OUTFIL` for writing on a certain external file. The name of that external file name is the default (system, dependent, `"TEXTOUTDVI"` or something like that) if no input file was ever mentioned, that is, if all user input was given on the terminal. Otherwise, it is a name related in some form with the name of the first mentioned input file.

### DVI

Packs a byte into a quarter word position of a single word buffer `DVIWORD.INT`, and outputs this word to `OUTFIL` when filled.

### INTOUT

Breaks an integer into four bytes and outputs these to `OUTFIL`.

Detailed Specifications of Procedures in the T<sub>E</sub>X/PASCAL  
Compilation Unit, SYSDEP

by

C. L. Lawson

I. Zabala

M. Díaz

Stanford University, January 28, 1981\*

\*This document is an enlarged and revised edition of a paper with the same name, published by C. L. Lawson; Computing Memorandum No. 467, September 15, 1980, Jet Propulsion Laboratory, California Institute of Technology.

## Contents

	Page
Introduction . . . . .	1
APPNDREAL . . . . .	1
APPNDSTRING . . . . .	1
APPENDTONAME . . . . .	2
CHANPTR . . . . .	2
CLOSE . . . . .	2
CLOSEOUT . . . . .	2
DECLAREOFIL . . . . .	3
DVI . . . . .	3
EOFCHAN . . . . .	3
FORCEBUFFEROUT . . . . .	3
GETCHAN . . . . .	3
GETFIRSTLINE . . . . .	4
GETTABLESIZES . . . . .	4
INCHTER . . . . .	4
INTDELIMTB . . . . .	4
INTEQTB . . . . .	5
INTFILENAME . . . . .	5
INTFMEM . . . . .	5
INITHYPHENTB . . . . .	6
INITPAGETB . . . . .	6
INITSECONDMEM . . . . .	6
INITSTRINGS . . . . .	6
INITSYSDEP . . . . .	6
INLN . . . . .	7
INLNTERR . . . . .	7
INTOUT . . . . .	7
OUTCHERR . . . . .	7
OUTCHTER . . . . .	8
OUTLNERR . . . . .	8
PRINT . . . . .	8
PRINTFILENAME . . . . .	8
PRINTINT . . . . .	8
PRINTLN . . . . .	9
PRINTOCTAL . . . . .	9
PRINTREAL . . . . .	9
PRODUCESTRING . . . . .	9

---

READFONTINFO . . . . .	10
RELEASE . . . . .	10
RSETFILE . . . . .	10
RWRITEFILE . . . . .	11
SCANFILENAME . . . . .	11
SENDCH . . . . .	11
SENDLN . . . . .	11
SENDSTARTED . . . . .	12
SETTABLESIZES . . . . .	12
TRACELINE . . . . .	12
WRITEDELIMTB . . . . .	12
WRITEEQTB . . . . .	12
WRITEFMEM . . . . .	13
WRITEHYPHENTB . . . . .	13
WRITEPAGETB . . . . .	13
WRITESECONDMEM . . . . .	13

Detailed Specifications of Procedures in the TeX/PASCAL  
Compilation Unit, SYSDEP**Introduction**

The PASCAL compilation unit, SYSDEP, contains some fifty PASCAL procedures; i.e., PROCEDURES or FUNCTIONS. The purpose of this document is to give complete program specifications for each of these procedures.

The procedures are listed in alphabetical order. Each procedure is identified by name, the names and types of all formal parameters, and the type of its result if it is a FUNCTION.

Some information is given as to the purpose of each procedure, the method used in the procedure, the purpose of each argument and FUNCTION result, and restrictions (such as valid ranges of input parameters).

The author does not have any depth of knowledge about the code being described. This is offered as a first cut at a document that would be more complete and correct.

For more details, see the complementary document called "Brief Functional Characterizations of the Procedures in the TeX/PASCAL Compilation Unit SYSDEP", which gives a higher level perspective on the functional groupings of these procedures, and describes more accurately the purpose of each procedure. Here we shall content ourselves with shorter functional descriptions.

**APPNDREAL**

Integer function with the following two parameters:

<i>Name</i>	<i>Type</i>
D	INTEGER
S	REAL

Appends the real number S to string D. The rules for D and the returned value are the same as those in APPNDSTRING below. Before appending, the real number S is converted to a string of five characters that has a decimal point in the fourth position.

**APPNDSTRING**

Integer function with the following two parameters:

<i>Name</i>	<i>Type</i>
D	INTEGER
S	REAL

Appends the real number S to string D. If  $S \leq \text{FILNAM}(0)$  then the corresponding string can be found in the FILENAME array, otherwise it is an ordinary string in STRNGPOOL. If  $D > \text{FILNAM}(0)$  this procedure does not append but instead creates a new string at the end of the STRNG and STRNGPOOL arrays and fills it with the contents of the source. In any case, the function returns an identifier for the string as required by PRODUCESTRING.

**APPENDTONAME**

Function of type INTEGER with two parameters.

<i>Name</i>	<i>Type</i>
CMMD	INTEGER
CH	INTEGER

This is related to procedure SCANFILENAME.

**CHANPTR**

Function of type ASCIICODE with one parameter.

<i>Name</i>	<i>Type</i>
ID	INTEGER

The integer ID must be in the range [1..6]. It selects one of ICHAN1 through ICHAN6 and executes CHANPTR:=ORD(ICHAN<sub>x</sub>↑).

This returns the ordinal value of the current input character in ICHAN<sub>x</sub>.

**CLOSE**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
STREAM	INTEGER

The integer STREAM must be in the range [0..9]. It selects one of OCHAN0 through OCHAN9 and executes RESET(OCHAN<sub>x</sub>).

According to page 12 of the Stanford documentation this has the effect of closing and saving the external file previously associated with OCHAN<sub>x</sub> for writing.

**CLOSEOUT**

Procedure with five parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
	DVIDYTECNT	INTEGER
	LASTPAGEPTR	INTEGER
	MAXPAGEHEIGHT	REAL
	MAXPAGEWIDTH	REAL
Var	PARBASE	FBASEARRAY

This procedure is called by TeX just before stopping. It writes the postamble information to file UTFIL by use of the procedure DVI.

This information appears to be partially system dependent.

**DECLAREOFIL**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
FYL	INTEGER

Initializes the output on the file indicated by FYL; that is, sets up correspondence between the internal file name OUTFIL and the external name.

The file OUTFIL will be opened using REWRITE.

**DVI**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
BYTE	INTEGER

Packs integer BYTE into a quarter-word position in DVIWORD.BIT as selected by the global variable BYTENUM which is in the range [1..4]. BYTENUM is incremented.

When DVIWORD.INT is filled it is output to the file OUTFIL.

**EOFCHAN**

Function of type BOOLEAN with one parameter.

<i>Name</i>	<i>Type</i>
ID	INTEGER

The integer ID must be in the range [1..6]. It selects one of ICXHAN1 through ICHAN6 to be tested, returning the value EOF(INCHANx).

**FORCEBUFFEROUT**

Procedure with no parameters.

This procedure executes the single nonstandard procedure, BREAK(TEROUT).

**GETCHAN**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
ID	INTEGER

The integer ID must be in the range [1..6]. It selects one of ICHAN1 through ICHAN6 to be accessed using GET(ICHANx).

**GETFIRSTLINE**

Function of type **BOOLEAN** with six parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
	FYL	INTEGER
Var	BUFFER	BUFFER
Var	BUFPTR	INTEGER
Var	BRCHAR	ASCII CODE
Var	EOFF	BOOLEAN
Var	PAGE	INTEGER

The integer FYL must be in the range [1..6]. It selects one of the files ICHAN1 through ICHAN6.

This procedure is highly system specific. It is used to skip system header stuff at the beginning of file number FYL. It also prints a "page" number which relates to the fact that DEC-10 files are partitioned into "pages".

The value of GETFIRSTLINE is set to true if the function is successful.

**GETTABLESIZES**

Function of type **BOOLEAN** with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	SIZESTABLE	SIZESARRAY

Opens a file for input with internal name TBLFIL and external name "TEXINITBL".

Reads twelve integers into the array SIZESTABLE. Sets GETTABLESIZES:=TRUE if no end of file is encountered.

**INCHTER**

Function of type **ASCII CODE** with no parameters.

This function reads one character from the terminal and returns its ordinal value. It first executes BREAK(TROUT) and RESET(TERIN,'FOOBARTY', , , ). After reading a character it skips to a carriage return.

**INITDELIMTB**

Procedure with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	DELIMTABLE	DELIMARRAY

Reads-in original contents of delimiters table.



INITFMEM

5

INTEQTB

<i>Name</i>	<i>Type</i>
EQTB	EQTBARRAY
HASH	HASHARRAY
HHEAD	HHEADARRAY
HASHPAR	INTEGER
HASHSEND	INTEGER

Reads-in original contents of equivalents and related tables.

INITFILENAME

<i>Name</i>	<i>Type</i>
FNUM	INTEGER

The function INITFILENAME initializes the data structures that will be used by APPENDTONAME to obtain the name of a file and open it.

If its argument lies in the range  $0 \dots \text{MAXFNT} + 1$ , the file is a font. The name should be allocated in the corresponding place in the first  $\text{NFONTS} + 1$  entries of FILENAME and default name conventions for font names apply (Only  $0 \dots \text{MAXFNT}$  are used for fonts properly, the extra entry is used for comparisons in DEFINEFONT.) If FNUM is in the range  $\text{NFONTS} + 1 \dots \text{NFONTS} + 10$ , the file contains an output send stream. A negative FNUM indicates an input file whose name must be allocated in the top position of FILENAME.

INITFMEM

Procedure with nine parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	FMEM	FMEMARRAY
Var	WDBASE	FBASEARRAY
Var	HTBASE	FBASEARRAY
Var	DPBASE	FBASEARRAY
Var	LGBASE	FBASEARRAY
Var	MSBASE	FBASEARRAY
Var	PARBASE	FBASEARRAY
Var	FONTINFO	FNTINFOARRAY
Var	FMEMPTR	INTEGER

Inputs the font memory from file TBLFIL.

**INITHYPHENTB**

Procedure with six parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	READOUTVARIABLE	TABLEREADOUTTYPE
Var	EXCEPTABLE	EXCPARRAY
Var	EXCEPHYPH	EXCPHYARRAY
Var	SUFFIX	SUFFIXARRAY
Var	PREFIX	PREFIXARRAY
Var	BTABLE	BARRAY

Inputs hyphenation tables from file TBLFIL.

**INITPAGETB**

Procedure with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	PAGEMEM	PAGEMEMARRAY

Inputs the page memory from file TBLFIL.

**INITSECONDMEM**

Procedure with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	MEM	MEMARRAY

Inputs from file TBLFIL to MEM[J].INT for J:=SECONDMEM to MEMSIZE.

**INITSTRINGS**

Procedure with no parameter.

Reads the ASCII table from external file "ASCII TBL", and the printable strings from external file "STRINITBL", into the array STRINGPOOL. Reading is done using ICHAN1, which is reset or each of these two external files.

Pointers into STRINGPOOL are set in the array STRNG.

**INTSYSDEP**

Procedure with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	FOURBYTESIZE	INTEGER

Initializes the values of FOURBYTESIZE, FOURBYTEMASK, FILPTR, BYTENUM, DVIWORD.INT. Clears the array FILENAME to NULLs.

Opens terminal for ASCII output. Internal name is TEROUT, external name is "FOOBARTTY".

Opens error file for output. Internal name is ERRFIL, external name is "ERRORSTEM".

## OUTCHERR

7

## INLN

Procedure with five parameters.

Var.?	Name	Type
	FYL	INTEGER
Var	BUFFER	BUFFIT
Var	BUFPTR	INTEGER
Var	BRCHAR	ASCII CODE
Var	EOF	BOOLEAN

The integer FYL must be in the range [1..6]. It selects one of ICHAN1 through ICHAN6 and reads from the selected file into the array BUFFER up to a carriage return, form feed, end of file, or the capacity limit of BUFFER.

On return BUFPTR will be the index of the last item in BUFFER, BRCHAR will be a copy of BUFFER[BUFPTR], and EOF will be true if an end of file was encountered.

## INLNTER

Procedure with five parameters.

Var.?	Name	Type
Var	BUFFER	BUFFER
Var	BUFPTR	INTEGER
Var	BRCHAR	ASCII CODE

Reads text into the array BUFFER from the terminal, up to a carriage return or the capacity of BUFFER. It stores ordinal values of the characters in BUFFER.

Sets BUFPTR to index the last value stored into BUFFER.

Sets BRCHAR as a copy of BUFFER[BUFPTR]. This will be either the ordinal value of carriage return or else a zero indicating the capacity of BUFFER was reached.

## INTOUT

Procedure with one parameter.

Name	Type
I	ASCII CODE

Breaks the integer *I* into four quarter-wordbytes and outputs the four bytes to OUTFIL using four calls to the procedure DVI.

## OUTCHERR

Procedure with one parameter.

Name	Type
C	ASCII CODE

This procedure outputs the character whose ordinal number is *C* to the error file, ERRFIL.

**OUTCITER**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
C	ASCHCODE

This procedure outputs the character whose ordinal number is *C* to the terminal, using the internal file name, TEROUT.

**OUTLNERR**

Procedure with no parameters.

This procedure outputs a carriage return and line feed to the error file using the internal file name, ERRFIL.

**PRINT**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
MES	INTEGER

Outputs string number MES in STRINGPOOL to the terminal, TEROUT, and to the error fill, ERRFIL.

**PRINTFILENAME**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
FYL	INTEGER

This procedure prints the name of the file whose integer identifier is FYL. The procedure PRINT is used to output to the terminal and the error file. The terminal printing is forced to complete by use of procedure FORCEBUFFEROUT.

The file name is fetched from FILENAME[FYL].

**PRINTINT**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
N	INTEGER

Converts te integer *N* to decimal digits from printing and outputs the decimal digits, and a minus sign if  $N < 0$ , to the terminal and the error file using procedure PRINT.

## PRODUCESTRING

9

## PRINTLN

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
MES	INTEGER

Outputs a carriage return and line feed to the terminal, TEROUT, and the error file, ERRFIL, and then uses PRINT to output message number MES from STRINGPOOL to these same two destinations.

## PRINTOCTAL

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
N	INTEGER

Converts the integer  $N$  to a string of twelve octal digits and outputs these to the terminal and the error file using procedure PRINT.

## PRINTREAL

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
X	REAL

Prints the real number  $X$  as an optional sign followed by decimal digits, a period, and four more decimal digits. This string is output to the terminal and the error file using the procedure PRINT.

## PRODUCESTRING

Procedure with two parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
X	INTEGER	INTEGER
Var	NAMESTRING	ASCHSTRING

If  $C \leq -2$ , this procedure returns the file name associated with font number  $C$ . The file name will be placed in NAMESTRING terminated by a NULL.

If  $C > -2$ , this procedure moves string number  $C$  from STRINGPOOL into NAMESTRING, with a terminating NULL.

**READFONTINFO**

This is an integer function that has the following parameters:

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
	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 integer FYL is used as an index in the various array parameters to establish the destination of this information.

**RELEASE**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
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.

<i>Name</i>	<i>Type</i>
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.

**RWRITEFILE**

Procedure with four parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
	ID	INTEGER
	FNAME	CHAR9
	FDIRECTORY	INTEGER
	FDEVICE	CHAR6

The integer *ID* must be in the range [0..9]. It selects one of OCHAN0 through OCHAN9 to be opened for output and associates it with FNAME, FDIRECTORY, and FDEVICE.

**SCANFILENAME**

This function has only one parameter.

<i>Name</i>	<i>Type</i>
FYL	INTEGER

It belongs to the main module of TEX/PASCAL, but because file names are very system dependent, the parsing is done in the system dependent module. Refer to the Functional Description of SYSDEP.

**SENDCH**

Procedure with two parameters.

<i>Name</i>	<i>Type</i>
STREAM	INTEGER
C	ASCII CODE

The integer *STREAM* must be in the range [0..9]. It selects one of OCHAN0 through OCHAN9 and outputs the character CHR(C) to file OCHAN<sub>x</sub>.

**SENDLN**

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
STREAM	INTEGER

The integer *STREAM* must be in the range [0..9]. It selects one of OCHAN0 through OCHAN9 and outputs a carriage return and line feed to OCHAN<sub>x</sub>.

**SENDSTARTED**

Function of type **BOOLEAN** with one parameter.

	<i>Name</i>	<i>Type</i>
	<b>STREAM</b>	<b>INTEGER</b>

The integer **STREAM** must be in the range [0..9]. It selects one of **OCHAN0** through **OCHAN9** and executes **SENDSTARTED:=EOF(OCHANx)**. The result is true only if this output stream has been started and not closed.

**SETTABLESIZES**

Procedure with no parameters.

This procedure opens a file for output with internal name **TBLFIL** and external name "TEXINITBL". It writes twelve numbers to this file giving sizes of various tables.

**TRACELINE**

Procedure with four parameters.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	<b>BUFFER</b>	<b>BUFFER</b>
Var	<b>BUFPTR</b>	<b>INTEGER</b>

This procedure prints the line in **BUFFER** to the terminal, and the error file, and then reads a character from the terminal. A carriage return means leave **BUFFER** as it is. Anything else means read a new line from the terminal into **BUFFER**.

**WRITEDELIMTB**

Procedure with one parameter.

	<i>Name</i>	<i>Type</i>
	<b>DELIMTABLE</b>	<b>DELIMARRAY</b>

Outputs the delimiters table to **TBLFIL**.

**WRITEEQTB**

This procedure has the following parameters:

<i>Name</i>	<i>Type</i>
<b>EQTB</b>	<b>EQTBARRAY</b>
<b>HASH</b>	<b>HASHARRAY</b>
<b>HHEAD</b>	<b>HHEADARRAY</b>
<b>HASHPAR</b>	<b>INTEGER</b>
<b>HASHSEND</b>	<b>INTEGER</b>

Outputs the parameter arrays and variables to file **TBLFIL**.



## WRITEFMEM

Procedure with nine parameters.

<i>Name</i>	<i>Type</i>
FMEM	FMEMARRAY
WDBASE	FRASEARRAY
HTBASE	FBASEARRAY
DPBASE	FBASEARRAY
LGBASE	FBASEARRAY
MSBASE	FBASEARRAY
PARBASE	FBASEARRAY
FONTINFO	FNTINFOARRAY
FMEMPTR	INTEGER

Write the font memory to TBLFIL.

## WRITEHYPHENTB

Procedure with six parameters.

<i>Name</i>	<i>Type</i>
READOUTVARIABLE	TABLEREADOUTTYPE
EXCEPTABLE	EXCPARRAY
EXCEPHYPH	EXCPHYARRAY
SUFFIX	SUFFIXARRAY
PREFIX	PREFIXARRAY
BTABLE	BARRAY

Outputs the parameter arrays to the file TBLFIL.

## WRITEPAGETB

Procedure with one parameter.

<i>Name</i>	<i>Type</i>
PAGEMEM	PAGEMEMARRAY

Outputs the parameter array to TBLFIL.

## WRITESECONDMEM

Procedure with one parameter.

<i>Var.?</i>	<i>Name</i>	<i>Type</i>
Var	MEM	MEMARRAY

Outputs MEM[J].INT for J:=SECONDMEM to MEMSIZE to the file TBLFIL.

## OUTPUT DEVICES: A NEW COLUMN

Barry Doherty

To aid the development of output device interfaces, we would like to collect and report information on work in progress on interfaces. It would greatly help both those who are considering installing an output device and those who are working on interfaces to have information on who is working on interfaces for which devices. If you are involved in such a project, please send TUGboat the following information for this column:

- host computer (and operating system);
- output device;
- status of the work;
- your name and phone number and your organization's name and address.

Please send all information to TUGboat. We'll start reporting it with the next issue.

\* \* \* \* \*

### Site Reports

\* \* \* \* \*

#### STATUS OF T<sub>E</sub>X ON THE STANFORD IBM 370/3033 SYSTEMS

Eagle Berns

After obtaining a copy of the IBM PASCAL/VS system, work was begun in conjunction with Luis Trabb-Pardo and Ignacio Zabala, on the implementation of T<sub>E</sub>X at Stanford. After a bit of debugging of both the PASCAL/VS and the T<sub>E</sub>X systems, I was able to get as far as generating a device independent output file from T<sub>E</sub>X which was sent on tape over to the Computer Science Department. The next day I received back my output from Luis with a note saying "looks good!"

However, the version of T<sub>E</sub>X which was working was far from being the end product. This was due mainly to the fact that the version of PASCAL being used permitted only 8-byte REAL variables. T<sub>E</sub>X, on the other hand, required only 4-byte REALs. Since these reals overlaid some integer 4-byte variables in an array which is 32K words long I was essentially using 128K bytes more than necessary. (This number was even larger in total as there were other reals in the system.) The PASCAL/VS group came through, and on a pre-release of the next version of PASCAL/VS, the variable type SHORTREAL existed as a 4-byte real variable. However, due to other changes, my original running version of T<sub>E</sub>X no longer would run! (Sound

familiar!) Along with this came other smaller problems, including the problem of making PASCAL run in an interactive mode in ORVYL (Stanford's timesharing system). My primary objective was to get PASCAL/VS up and running for the Stanford community. Hence, this is where the main burst of activity is taking place. Of course, T<sub>E</sub>X will be the first major program to be run using interactive PASCAL. Projections about when a working version of T<sub>E</sub>X will be available run in the neighborhood of the end of February to mid-March (1981), although no fixed commitment is being made at this time.

Some initial looking into the hardware to be used for printing T<sub>E</sub>X at Stanford has begun, but this is a separate project and nothing substantial can be said at this time.

Those interested in information on the availability of the T<sub>E</sub>X implementation at C.I.T. can send me a letter (no phone calls—I have a terrible time not losing phone messages) with an address to forward information to, and when firm dates for its availability can be made, a letter will be sent out.

\* \* \* \* \*

#### AMS SITE REPORT

Barry Doherty and Barbara Beeton

AMS equipment consists of a DEC 2060 running under TOPS-20, a Varian 9211 printer/plotter, an Alphatype CRS and a Florida Data dot-matrix printer (the latter located at the Math. Reviews office in Ann Arbor). Both the Varian and the Florida Data are driven by Monolithic Z80s configured according to the Stanford specification.

We have been using the SAIL version of T<sub>E</sub>X for well over a year. Most effort so far has been devoted to converting various administrative publications (*Combined Membership List*, *Administrative Directory*, *Catalogue of Publications*) and the issue indexes for *Math. Reviews* and *Current Mathematical Publications* to T<sub>E</sub>X format. Certain regular departments of the *Notices* have been converted to T<sub>E</sub>X, and an experiment is underway to typeset material for the *Abstracts*, using AMS-T<sub>E</sub>X. Some of the T<sub>E</sub>X files are program-generated and maintained, while others have been created and maintained manually.

Varian output is used primarily for proofing, although some camera copy is still generated on it (primarily due to lack of a complete set of fonts for the Alphatype); we expect that most camera copy will soon be produced by the Alphatype. The Florida Data is set up to produce output identical

in size to that from the Varian, using fonts constructed especially for this purpose; a compatible set of fonts is being developed by the Ann Arbor staff (an article on this project will appear in a later issue of TUGboat). The Florida Data, with its ability to print on card stock and preprinted forms, will be used to generate records (via T<sub>E</sub>X) for Math. Reviews; the Florida Data is also being used as a line printer.

Both the Varian and the Florida Data are currently run by separate spoolers, although we have designed an integrated spooling system and are in the process of implementing it.

We expect to extend our use of T<sub>E</sub>X to mathematical journals, relying to a great extent on A<sub>M</sub>S-T<sub>E</sub>X, perhaps beginning some time this year. The timing of the transition depends on the completion of necessary, but currently nonexistent fonts, particularly of mathematical symbols and non-roman alphabets.

We also have a strong interest in METAFONT, and have managed to get a version (modified from the Stanford original) running on the DEC20, although it is currently unable to prepare fonts for the Alphatype. So far we have used METAFONT mainly to generate fonts for the Florida Data, using Knuth's existing Computer Modern designs with different size parameters; a similar technique has been used to generate some Varian fonts in sizes that were unavailable. We are also developing a special compact ("telephone book") font based on CMSS6, which in turn was adapted from CMR6 and CMSS10.

We are hoping to develop full METAFONT capability, so that we can produce in-house any new fonts needed for full journal production.

\* \* \* \* \*

## THE STATUS OF VAX<sup>†</sup>/T<sub>E</sub>X AT BROWN

Janet Incerpi

At Brown University we are implementing Pascal T<sub>E</sub>X using a virtual memory approach without any special purpose hardware to drive the printer/plotter. Pascal T<sub>E</sub>X has been compiled under Berkeley UNIX<sup>‡</sup> on a VAX 11/780. Also, a scan conversion program that sends output to a Benson-Varian 9211 has been completed.

Those who are working on implementing Pascal T<sub>E</sub>X know that there is more involved than just getting a very large Pascal program to run. The four

components necessary for a working system are the Pascal T<sub>E</sub>X program, font metric files, font files and a scan conversion program. The font metric files contain character measurements (e.g. height, width and depth), ligature and kerning information, and various information used in setting mathematical formulas. T<sub>E</sub>X uses the font metric files to do the typesetting. T<sub>E</sub>X outputs a description of the pages being typeset by specifying font and character placement. The font files contain the actual bit rasters (dot-by-dot pictures) for the characters. A scan conversion program is necessary to convert from the ASCII characters in the file output from T<sub>E</sub>X to the bit raster format (from the font files) needed to print the characters. The scan conversion program uses the output of T<sub>E</sub>X and the bit rasters from the font files to produce a large bit array which defines the printed page.

### Using Virtual Memory for T<sub>E</sub>X

In standard implementations the T<sub>E</sub>X output, called a DVI file, is passed through a program, DVIVER, which creates an intermediate file. This is passed to a second program, VERSER, which communicates through a low-speed line with a microprocessor that does the scan conversion to a printer/plotter. Our system does not use a microprocessor; instead it uses the direct memory access (DMA) facility of the printer/plotter interface when doing the scan conversion.

On our system, the DVI file is used as input to a position-sort program that is similar to DVIVER. This generates an intermediate file that contains the necessary packed information, including the character's horizontal ( $x$ ) position, the change in vertical ( $y$ ) position, the font to use, the character and the part number. The part number is necessary if the character is too large and therefore must be broken into parts. This intermediate file is constructed a page at a time with each page sorted on the  $y$  coordinate before the information is placed in the file. This file is used as input to the scan conversion program which sends the output to the printer/plotter.

The virtual memory facility provided by Berkeley UNIX enables us to handle a large number of font files, while requiring only the pages used to be in main memory. The way we handle the large number of font files is to have an array of font file pointers. Space is allocated for a font file only if that font number is to be used. Although the space is allocated, only those pages of the font file which are referenced using the font file pointer are actually read in. It is no longer necessary to switch font files in and out of memory when the space allocated has been used. This approach does, however, make

<sup>†</sup>Insiders pronounce the X of VAX as an "x", not as a Greek chi, so that VAX rhymes with the word "hacks".

<sup>‡</sup>UNIX is a trademark of Bell Laboratories.

the system harder to export to non-virtual memory machines. For example, with the VNT font files, which are used for output to the printer/plotter, the pages containing the character information (i.e., pixel height, width, etc.) are read in, as well as any pages that contain the bit rasters of any characters that are used from the font. The scan conversion program uses the information from the intermediate file and the VNT font files to send bits to the printer/plotter. It uses a wraparound buffer into which it places the bit rasters for characters or vertical and horizontal rules. When the buffer must be emptied (if a character is too tall or a change in  $y$  is too large), the scan conversion program writes the appropriate rows to the printer/plotter. To minimize the number of transfers, each write sends the maximum number of rows, taking advantage of the printer/plotter interface's DMA feature.

When the scan conversion program gets the bit rasters from a VNT font file, it is important for it to know that each row of the raster starts on a word boundary. Any bits of the last word in a raster row which are not used are zero. The scan conversion program skips over these trailing zeroed bytes since the buffer is always cleared after its contents are written out.

#### Pascal and Word-Size Problems

As mentioned earlier, we have compiled Pascal  $\TeX$ . This was only possible after manually fixing the case statements in the  $\TeX$  source code since our Pascal adheres to the standard of no default (OTHERS) case. The remaining problems that must be fixed to get  $\TeX$  running are system dependent.

One such problem is the conversion of the 36-bit-word oriented TFX files, the  $\TeX$  font metric files, to our 32-bit machine.  $\TeX$  uses these files to do the typesetting. We obtained both the VNT font files and the TFX font metric files from AMS, which runs  $\TeX$  on a 36-bit machine. The transition onto our 32-bit machine was straightforward for VNT files, since these files only use the low-order 32 bits of each word. Unfortunately TFX files are not as easily adaptable. The TFX file descriptions we have seen use all 36 bits of a word, but in two sources there are references to possible solutions.

The two possible solutions deal with ignoring bits in the TFX file words. The words in TFX files are divided into various fields. Two such fields are the device-width (6 bits) and the ligature (9 bits) fields. One source claims the device-width field is not used, while another says that only 5 bits are necessary for the ligature field. Either way our problem would be solved; unfortunately we do not know whether these are really two solutions to the problem. Once we

have our TFX files set up we need only be concerned about  $\TeX$ 's accessing them.

We had difficulty making variant records in Pascal as clean as in standard Pascal  $\TeX$ , because our current compiler makes packing and unpacking information in file words more difficult. For example, in Pascal  $\TeX$ , to output a word to the DVI file, four bytes are packed into a word using a variant record which is one of two possible cases. The record is either four scalars (ranging from 0 to 255) or is an integer, so the  $\TeX$  procedure, *dvi*, uses scalars (bytes) to pack the information and then writes the integer to the DVI file.

However, complications arise because our Pascal implementation does not overlay the scalars and the integer as one might expect. For our compiler, we found a way to correct this but the solution is far from ideal. We expect the same type of revisions to be necessary to access the TFX file information.

The other changes that must be made to Pascal  $\TeX$  are those dealing with the file system and user interaction. These routines will be rewritten.

#### The Status of Scan Conversion

Though we haven't yet run Pascal  $\TeX$  on the VAX (to generate DVI files) we have tested the scan conversion program. (We were able to obtain DVI files from AMS through Sail  $\TeX$ ; in fact, this paper has been printed using our VAX scan conversion program.) The scan conversion program generated the bits for the correct output. Therefore we do have both the position-sort program and the scan conversion program working. Both of these programs are written in C because it allows easy bit manipulation and the use of virtual memory facilities. Later, we will make these programs more transportable.

#### Future Plans

Our work will be concentrated in three major areas once we have Pascal  $\TeX$  up and running. First, we plan a detailed performance study to help evaluate our implementation approach for the scan conversion program (i.e., relying on virtual memory as opposed to special purpose hardware). Second, we intend to get Pascal  $\TeX$  working under different system environments, including VAX/VMS, various IBM systems and implementing scan conversion for experimental graphics and printing devices. Third, we plan to eventually build high-level facilities on top of  $\TeX$  for document production.

Please direct any questions or suggestions to either Prof. Robert Sedgewick or the author, at The Department of Computer Science, Box 1910, Brown University, Providence, RI 02912.

\* \* \* \* \*

**REPORT FROM THE NORTH STAR**

— or —

**T<sub>E</sub>X AT THE UNIVERSITY OF MINNESOTA**

T. D. Hodge

In December, 1980, we received a new tape copy of T<sub>E</sub>X-in-Pascal from Stanford. Working with this tape, we have gotten the T<sub>E</sub>X preprocessor to read the text file and have partially converted the font files (TFX files) on our CDC Cyber 74.

The main problem we have encountered is the fact that the TFX files contain DEC10 floating point numbers. Ignacio Zabala has promised us that the font files on the next version we receive will all be in integer.

Meanwhile, we will go ahead with the font files we have and try to get the preprocessor and T<sub>E</sub>X itself to produce workable output.

Initially, we expect to run T<sub>E</sub>X as a batch program because of its apparent large size. We hope to have more specifics about size in the CDC version and about other factors by the next T<sub>E</sub>X Users Group meeting in the spring.

Minneapolis, 30 January 1981

\* \* \* \* \*

**Editor's note:** The following item has been reproduced directly from copy submitted by the author.

**T<sub>E</sub>X IS AVAILABLE  
FOR UNIVAC 1100 SYSTEMS**

Ralph Stromquist  
1100 T<sub>E</sub>X Coordinator

The Academic Computing Center at the University of Wisconsin-Madison (MACC) has implemented T<sub>E</sub>X on a Univac 1100/82 computer using the UW-Pascal compiler. Testing had been limited by lack of a typesetter until the last two weeks, but output to a line printer and initial typesetter testing indicate that only minor problems remain.

We have tested most of T<sub>E</sub>X's features successfully: font changing, macro definitions, ligatures, footnotes, fractions, exponents, subscripts, variable size parentheses, matrices, alignments -- to name a few. We do not yet have extension fonts, so have not extensively tested mathematical typesetting. Tests on uncomplicated documents show T<sub>E</sub>X uses about 0.14 seconds of CPU time per 6.5 by 9 inch page with memory usage of 119,000 words.

Aside from being the first Univac implementation of T<sub>E</sub>X, our site is also unusual because we have a new Compugraphic 8600 phototypesetter. The 8600 produces high-quality 1300 scan lines per inch output. The model we have can hold about 100 digitized fonts on disk storage. The 8600 is operated offline via a magnetic tape drive. This has the advantage of easy control and backup if reruns are necessary for any reason.

T<sub>E</sub>X-1100 is available for distribution. Documentation is currently in preliminary form, but should be adequate for installation. Our package includes T<sub>E</sub>X, a font preprocessor program that converts readable font descriptions into T<sub>E</sub>X font files, our program for line printer proofing, the 8600 driver program, useful macros, and documentation. Although we don't expect most installations to have an 8600, our driver should still be useful as a model or framework.

To divide part of the cost of T<sub>E</sub>X-1100 development, distribution and maintenance among the sites using the package, we are charging \$500.00 for the package described above. The charge also covers program maintenance by MACC with distribution of enhancements and updates for a period of one year. We will also do limited telephone and written consulting.

The maintenance support may be continued after the first year by paying an annual fee of \$400.00. If you have questions concerning T<sub>E</sub>X-1100, contact

Ralph Stromquist  
MACC  
1210 W. Dayton St.  
Madison, WI 53706  
(608) 262-8821

Orders can be placed directly with the MACC Program Librarian at the above address.

This report was typeset by T<sub>E</sub>X on the Univac 1100/82.

\* \* \* \* \*

**REPORT ON THE USE OF T<sub>E</sub>X  
AT COMPUTAS A/S, NORWAY  
NORD 100 COMPUTER 16 bit "big mini"**

Helge Totland  
Computas A/S  
Box 310, N-1322 Høvik, Norway

Hello, all of you far away in the wilderness somewhere. This is a report on two people's (Tore Østensen and Helge Totland) exciting work with T<sub>E</sub>X. Well, perhaps we are not doing T<sub>E</sub>X work at all, some would say. Temporarily, we call what

we are working with FTEX (which means Formula part of  $\TeX$ ). We have a phototypesetting system (Nortext), and we are cooperating with the computer firm Norsk Data (they make "Nord" computers, sell Nortext systems and are funding the  $\TeX$  activity). What we missed most in Nortext is the ability to typeset formulas. While sometimes it might be advantageous to start a project from scratch, we started with the existing typesetting system Nortext (originally intended for newspaper work, the system might drive different phototypesetters). What we decided to do was to grab hold of the math part of  $\TeX$ , write formulas in  $\TeX$  code and other text in Nortext.

We started to work on this project last Spring (the "thinking" started November 1979). Our first aim was to have a compiler version of FTEX, with input in  $\TeX$  math code and output in Nortext code. Later we will integrate this compiler into Nortext, which will be turned into something  $\TeX$ -like when writing some appropriate "start code" (working like \$ or \$\$ in  $\TeX$ ). The first main user of this system will probably be CERN in Geneva (Switzerland) (Derek Ball). "Det Norske Veritas" (DnV) will also be a user, we expect. COMPUTAS A/S is the data division of DnV, which is an institution concerned with ship classification and other activities (so many technical documents are produced).

Our main source of information has, of course, been Donald Knuth's description of  $\TeX$ , which we have found very useful. Compared with other systems, the macro feature is especially useful. We had a few beneficial talks with Mike Bennett, while he was staying in Aarhus (Denmark). During our implementation we probably have invented the wheel (again) several times, but then at least we know what's going on. Alternatively, we would have to make rather significant changes in the Pascal code (which is not yet available anyway), if we should convert  $\TeX$  (or rather extract/convert the math part, change the output part and squeeze this into a 16-bit machine). Our FTEX compiler, by the way, is written in PLANC, which is a rather new systems programming language to Nord (not yet released—this has occasionally caused some new-language troubles).

In order not to stir up the  $\TeX$  wizards far, far away, we have tried to implement the math mode very close to what we imagine it works like at Stanford. As a side effect, this makes user documentation much less of a problem to us than it would otherwise be. We have a few special debug macros, which are not of any great concern. Our special macros begin normally with \*, which is not normally

defined as a letter (these macros are not available to users unless using `\chcode`). Example: `\*help` (list macro names on one or more levels), `\*list` (list macro definition), `\*status` (miscellaneous status info). One of the macros we have introduced in our system is `\kdef` (keydef), giving identical results to setting `\chcode` to 13 and defining a one-symbol macro (as described in the  $\TeX$  errata, TUGboat, October 1980, page 11). This makes definitions similar to `\def` and `\xdef`. It is of special interest to us as the Nortext terminals use 8-bit TET code and then more codes are available than in the 7-bit ASCII code.

We have a few other changes as well:

- cc means cicero (1 cc = 12 dd, ref. dimensions page 40 in  $\TeX$  manual. In Europe we often use cicero instead of pica (7% bigger (and better))).
- The category code table is extended (Special key, NUL and comment (end-of-line (category 5) is not (!) identical to comment (%))).
- Our internal precision is "only" 1/500 mm (to save space).
- We produce Nortext code instead of DVI-file (DVI-file would be a bit easier to make, but then we would still have one more chain the formulas had to pass before finally being typeset).
- The font files contain left- and right-space in addition (?) to height, depth and width (and we assume the symbol reference point is identical to an assumed focus point in centre of the lens—or similar on CRT's). We use 8 bits on each measure and multiply the measures to different sizes when in use.

The structure of the FTEX compiler is mainly as others have described in TUGboat Vol. 1 No. 1. We have a macro-expansion-section, a formula-tree-builder-section and a formula-tree-traverse-output-section. The first two of these sections work in parallel. We have not segmented the programs or made external intermediate files, as this would only delay the compiling process. When testing we have successfully produced code for typesetting rather complicated formulas (and flags, to test our rule code routine), but there are still features we have not implemented.

Thanks for the TUGboat newsletter. We enjoyed it, and the accompanying  $\TeX$  errata list will certainly be followed by some actions by us (where practically possible).

Oslo, 8 January 1981

\* \* \* \* \*

## Warnings &amp; Limitations

\* \* \* \* \*

**Disappearing Digits; Undisciplined Uppercase**

If a macro definition ends with any control sequence taking a `<number>` as a final argument, care must be taken to isolate the `<number>`, either by following it with a space (as in the definition of `\%` on page 151 of the `TEX` manual) or by burying the control sequence in a set of braces (the `\mix` definitions, page 167). For example,

```
\def\addr{ ... \penalty 1000}
... write to \addr 201 Charles St. ...
```

yields

```
... write to ... Charles St.
```

with no street address. The secret is in `TEX`'s interpretation of `<number>`, as described on page 34: `<number>` is a string of digits of any length.

In a similar-looking situation, Mike Spivak reports: "Complete havoc was wreaked on one of my macros when I typed

```
\chcode'046=12\newcontrolsequence
instead of
```

```
\chcode'046=12 \newcontrolsequence
Caveat chcoder!"
```

The following control sequences take `<number>` or integer arguments: `\char`, `\chcode`, `\chpar`, `\hangindent` (for and after options), `\penalty`, `\setcount` and `\spacefactor`.

`\uppercase` nicely ignores control sequences in its (token list), but not dimensions, so an `\hskip .5em` becomes `\hskip .5EM`, which `TEX` finds to be an ! Illegal unit of measure. One way of circumventing this problem is to anticipate it:

```
\uppercase{...\lowercase{\hskip .5em}...}
(\lowercase may not be interposed between the
skip and its (dimen).)
```

Barbara Beeton

\* \* \* \* \*

## Macros

\* \* \* \* \*

```
\title HOW TO PREPARE A FILE\cr
FOR PUBLICATION IN TUGboat\cr
```

```
\\Barbara Beeton\cr
American Mathematical Society\cr\end
```

An author writing an article for publication in `TUGboat` is encouraged to create it on a computer file and submit it on magnetic tape. Most of

`TUGboat` is composed by `TEX`, and makes use of the `AMS-TEX` macro package. `TUGboat` is thus a prototype of the journal-of-the-future as described by `TUG`'s Chairman, Dick Palais, in his message to readers of `TUGboat` #1.

Like any specialized journal, `TUGboat` has its own particular style, and a set of formatting macros has grown up to accommodate its needs. These can be used by an author to make his job (and that of the `TUGboat` production staff) easier. Depending on the author's local computer resource, he may or may not actually be able to `TEX` his file. Even if he can't, appropriately identifying material in his file by using these macros will serve to identify his intentions to the `TUGboat` staff `TEX`'er, with (we hope) salutary effect.

The first instruction is a simple one. A blank line or the control sequence `\par` indicates the beginning of a paragraph. This author prefers the blank line, since she finds densely packed and interminable text nearly impossible to read on either a terminal screen or a page of line-printer output.

Most `TUGboat` articles so far have consisted mainly of text, with occasional section headings, lists, addresses, and similar constructions. It is these that the `TUGboat` formatting macros try to cope with, not complicated tables or macro listings and other display material—these latter constructions will most likely be submitted by authors who have relatively great experience in scouting `TEX`'s byways, and any workable `TEX` coding will be acceptable for this kind of material. The present article is aimed mainly at the itinerant author who simply wishes to share his experiences with others in the `TEX` community.

Where practical, the macros will be illustrated rather than described. For example, the title of this article looks strange on purpose: the input format for a title is `\title (title)\(author, etc.)\end`, with the end of each line (as it is to be printed) marked by the instruction `\cr`. (Don't forget the space after a control sequence.)

```
\subtitle Subtitles\end
```

are also (rarely) desired. The "Disappearing Digits" article (page 53) is headed by a subtitle rather than a full title.

```
\lparhead SECTION HEADINGS,\lbrk
WHICH MAY BE MORE THAN\lbrk
ONE LINE LONG\end
```

and

```
\lparheadb Bold section headings\end
```

will look like the headings on pages 5 and 49.

`\parhead` RUN-IN HEADINGS, WHICH MAY OCCASIONALLY OCCUPY MORE THAN ONE LINE IF THE AUTHOR GETS CARRIED AWAY\end There is also a bold version, obtained with `\parheadb ... \end`.

`\parsub` Short, run-in subheadings.\end These also come in several flavors.

`\parsubp` (a) With parenthesized tags.\end The input for this automatically places parentheses around whatever precedes the first space in the argument string. `\parsubpr ... \end` differs only in that the text is roman. And if parentheses aren't wanted, but even spacing is desired between the tag and whatever follows, `\parsubr ... \end` will accomplish that, as in the numbered paragraphs on page 9.

`\hparhead` PARAGRAPH HEADINGS WITH HANGING INDENTS.\end are used only rarely, and may also come in boldface: `\hparheadb ... \end`. A more conventional use for hanging indent is

`\hparsubr`

- to list things (page 11) or\end

`\hparsubr`

2. to make a bibliography. In any case,\end

`\hparsubr`

\*\* `\hparsubr` \*\* is the ... \end is the input command which must delimit every tagged line at both ends.\end

`\endhpar` terminates a body of hanging indented material, and restores normal paragraphing.

An address can be set off from text for easy reference. Here's where you write if you wish to submit a TUGboat article on tape and have some questions that can't be answered by this article or by your local T<sub>E</sub>Xpert:

`\textaddr{`

Barbara Beeton\lbrk  
American Mathematical Society\lbrk  
P.O. Box 6248\lbrk  
Providence, R.I. 02940}

For people whose fingers get tired of typing the same thing over and over, a bunch of control sequences have been defined which just produce logos or strings of text:

<code>\TEX</code>	T <sub>E</sub> X
<code>\TUG</code>	T <sub>E</sub> X Users Group
<code>\tug</code>	TUG
<code>\TUB</code>	TUGboat
<code>\AMS</code>	American Mathematical Society
<code>\ams</code>	AMS
<code>\AMSTEX</code>	AMS-T <sub>E</sub> X
<code>\MF</code>	METAFONT
<code>\Pas</code>	Pascal
<code>\TIP</code>	T <sub>E</sub> X-in-Pascal
<code>\PT</code>	Pascal T <sub>E</sub> X

Remember that a space after a control sequence is gobbled up, to allow such constructions as "T<sub>E</sub>Xpert", and a control space `\_` should be used to make the space appear in the output. Authors are welcome to create control sequences for terms and phrases that they use frequently; anything of general usefulness may be cribbed by the TUGboat staff, and added to the list above.

When creating a file, it is usually a good idea to limit lines to the width of a terminal screen (usually 80 characters), even though your computer may allow much longer lines. This means, when you approach the right-hand end of the screen, hit the carriage return. (Also remember to insert a carriage return at the end of your file.) T<sub>E</sub>X arbitrarily assumes that no line should exceed 150 characters, and if one does, T<sub>E</sub>X dissects it after 150 characters, shoving the remainder onto the next (inserted) line. If a word is split at character 150, T<sub>E</sub>X recovers gracefully, but if a control sequence is split, results are unpredictable, and invariably incorrect. So for this reason, and also to make the processing of tapes routine, we ask you to limit lines to 80 characters, including the carriage return.

Finally, a word on magnetic tapes. TUGboat is produced at the AMS on a DEC 2060, which can read 9-channel magnetic tapes written at 800 or 1600 bpi, in a number of different, but well-defined, formats. To save time and avoid problems, tapes should be created by someone who has had experience in doing this. Most universities have a user service staff who can answer questions or even get the job done. If your local computer is a DEC10, the BACKUP program should be used to write the tape; if it's a DEC20, use DUMPER. Otherwise, the records written onto the tape should be 80 characters, blocked 10, and the ASCII character set should be used. A detailed list should accompany the tape, stating exactly what files are present. A form for submitting tapes to TUGboat is included with this issue; a copy of this form, with all the requested information filled in, should accompany each tape submitted. If you request it, your tape will be returned to you.

\* \* \* \* \*

Editor's note: The response to our request for information on macro packages and related software for publication in TUGboat has been encouraging. Our thanks to all.

This is your column. Your contributions and comments are needed. Deadline for the next issue is June 15, 1981.



T<sub>E</sub>X Macro Package

by

Max Díaz\*

This macro package, consisting of about 450 control sequences, is suited for most applications: memoranda, letters, papers, reports, thesis, books. Provisions exist for working with ease (one-character accents, without exceptions, for instance) in one of three languages at the turn of a switch: `\english`, `\espanol`, `\francais`.

It contains all standard simple features one would expect: paragraphing (enumerate, display, itemize, etc.), footnotes, annotations, comments, quotes, verses, etc. And, for papers and longer documents, it also includes handling of chapters and subsections (up to three levels), with automatic tables of contents and plates (figures, tables) and headings. There are many other useful macros, such as "no-fill" mode (for computer programs, say) and a collection of "Mickey Mouse macros" for glamorous effects.

Special care has been put to make output routines simple. For example, document's preambles may look like this (there are default settings for everything, of course):

```

\romannumbering{7}                % Start on page vii
\bothsides                        % Adjust for both-side printing

\setmargin{1.25}{1}               % Margin on odd pages=1.25", even=1"
\columnspanpage{4}{1.5}{.08}{6.25} % On 6.25" sheets, set 4 pages per sheet:
                                   % width 1.5", interpage spacing .08",
\setvsize{9}                      % and vertical size 9"

\onelineheading                   % Select style

```

The last command will select among some six possible types of page's style: this includes the headings, headlines, title pages and normal pages formats. The macros that set dimensions adjust these, and any other macros, automatically.

Given any page design, no matter how appealing you think it is, there exists someone who will utterly dislike the font used for subsections, or the § sign, or the spacing in an itemized paragraph (the author has typed or helped typeset about five thesis, several reports, and many letters). For that reason, most macros utilize "parameters" that can be redefined without digging deeply into the whole package to locate the site where a particular object is being processed.

The package is by no means "complete." At the time of this writing, index and "grafix-mode" macros are being written. Frugal documentation exists, but a better non-T<sub>E</sub>Xpert document is being composed. The macro package is totally compatible with the T<sub>E</sub>X manual; no standard control sequences are redefined, and no special extensions of T<sub>E</sub>X are required either (so far).

Macros are currently lodged in the S.A.I.L. machine, and a non-SAIL version exists at SCORE. To obtain them, send (ARPAnet users) a message to MMD@SAIL, or write to the author.

---

\*Mathematics Department/ Stanford University/Stanford CA 94305

## Anatomy of a T<sub>E</sub>X Macro Package

Arthur M. Keller

Stanford University

**Abstract.** A T<sub>E</sub>X macro package is described that produces various formats of output. Parts of the macro package are based on the basic and book formats described in the T<sub>E</sub>X manual as well as the format for the T<sub>E</sub>X manual itself.

### 1 Introduction

This macro package was developed over approximately a year and a half. It started when I first began to use T<sub>E</sub>X, and proceeded as I began to write macros that others found useful. Much of the development has been for a forthcoming book to be published by McGraw-Hill teaching PASCAL to students without a knowledge of computers.\* Further, the work of others has been borrowed and adapted in preparing these macros.

It is assumed that the reader is familiar with T<sub>E</sub>X. This documentation is intended to be read while perusing the code itself.

The macro package consists of a file called ARKTEX.TEX which should reside in the T<sub>E</sub>X system files area. This file refers to other files which are loaded if needed. This package was designed on the SU-AI system and uses the SAIL character set as described in the T<sub>E</sub>X manual. Others not using the SAIL character set may find it useful to change the \chcode's at the start of the file as well as some of the one character macros.

### 2 ARKTEX.TEX

The file ARKTEX.TEX is divided up into about twenty sections separated by horizontal rules. In the file, these are usually on separate pages (i.e., separated by control-L's).

#### 2.1 Standard Basic Stuff

This section consists primarily of text that appears in BASIC.TEX. The \chcode's should be changed as necessary for your system.

#### 2.2 Font Definitions and Related Macros

This section consists of macros for fonts of various fonts and sizes. The \chcode on the first line is to all © to be parsed correctly on this page because elsewhere © has \chcode of 13. The fonts on this page are primarily for eight, nine, and ten point typesetting. Other random fonts also exist. The \: macro has been redefined to save the font letter in \fontcode. This is used by the macros in "Definitions of Odd Characters" that produces output of different characters depending on which font is currently in use. Do not use the \usefont macro, but the \curfont macro may be used as an alternate to the \: macro.

The \loadfont macro is used to allow documents to use fonts that exist only on some systems to refer to these fonts symbolically. Ordinarily these fonts are not preloaded. However, a document attempting to use such a font for a particular output device for which the font exists may do so. For example, at Stanford, some of the fonts loaded by \loadfont exist only on the XGP or on the Dover but not on the Alphatype.

---

This work was supported in part by the T<sub>E</sub>X project under Prof. Donald E. Knuth and Dr. Luis Trabb-Pardo, and by a National Science Foundation Graduate Fellowship.

Author's address: Computer Science Dept, Stanford University, Stanford, CA 94305. ARPANET address: ARK at SU-AI.

\*The book will probably be titled *A First Course in Computer Programming Using PASCAL* and it will be available about January 1982.

The various sizes of type are referred to by the macros `\tenpoint`, `\ninepoint`, and `\eightpoint` to obtain ten, nine, and eight point type faces respectively. For each size there are the following fonts provided: roman (`\rm`), slanted roman (`\sl`), boldface roman (`\bf`), italic (`\it`), math italic (`\mi`), teletype (`\tt`), and symbol (`\sy`). In addition, small caps (`\sc`) is provided for ten point. Each size of type also includes definitions for the width of a digit (`\9`) as well as complete math mode information.

To start text in one of these sizes specify `\tenpoint`, `\ninepoint`, or `\eightpoint`. Customized macros for various point sizes may be constructed by redefining the macros `\usertenpoint`, `\userninepoint`, and `\usereightpoint`. However, if the most of the document is to be in that type size, say `\usertenpoint`, etc., instead at the start of the document. If you use `\startcode` and `\endcode`, `\fontsize` will be used to determine what font size to return to. This macro may be defined at any time prior to such usage, but it can be done easily by saying `\usertenpoint` if desired.

### 2.3 Definitions of Odd Characters

This section includes definitions that will allow characters such as `@` and `#` to be parsed correctly in any mode. Characters that have no other usage, such as `@` may appear without a preceding `\`, as they have a `\chcode` of 13. Characters that have other usages, such as `#`, must be preceded by a `\` if they are to appear as the character instead of being used for their standard purpose. However, because they are control sequences without an argument, you must put `\_` following them to avoid the space after them getting ignored.

Characters in teletype mode are fixed width characters. Therefore, the `\ttchar` macro takes a specified character and puts it in the desired size box so that the remainder of the line will line up.

The `\fontclassify` macro determines which mode or typestyle is being used and chooses the correct argument to emit. This allows `\#` to produce a `#` in the right font.

### 2.4 Redefinitions of One Character Macros

This section consists of redefinitions of one character macros so that they work in any mode. Users who redefine macros such as `\le` to print `≤` will also find the redefinitions useful. Macros starting with `\M` are defined to save the original definitions. Macros without an `M` are defined to work in or out of math mode. These original macros are redefined to match the new set. The exception is `\!` as it means different things in math mode than in non-math mode.

### 2.5 Make Some Math Things Work Anywhere

This group of macros works just like those in the previous section.

### 2.6 Page Numbering

The section on page numbering is rather complicated by the existence of macros to defer text. The pertinent macro for users is `\setpagecount` which sets the page number on the following page to the specified number. The page number on the current page is set to one less than that number. This mess is done because the author likes to put out an extra page describing what is going on whenever changing page numbers. In particular, output devices that do not put out header pages permit several users output to appear without intervening pages. Putting out your own separator page reduces the chance that your output will be misfiled. Lastly, the `\chapterbegin` macro in book format ejects the page first, so the author usually puts fixed garbage on the previous page. It's also a good place to put copyright notice if the file is going to be copyrighted. Most importantly, using `\settitle` of book format on the ejected page guarantees that the next page will have the correct headings.

### 2.7 `\output`, Style, and Format Routines

These macros are the heart of the claim of providing many formats of output. They fall into four

categories: overhead, output, style, and format routines. Overhead routines are used in many places and are obvious. These are `\normal`, `\resetsize`, and `\everyoutput`.

The output routines set the `\output` macro. Also, they should also set the macros `\normalhsize` and `\normalvsize`. See `OPLAIN.TEX` for the minimum required in an output routine.

Style routines set such things as paragraph spacing. See `OBLOCK.TEX` for the minimum required in an style routine.

Format routines are simply style and output routines in the same file or macro.

See the descriptions of the individual files for more information.

## 2.8 Footnotes

The footnote macro has gone through several generations. The latest one uses the `\botsep` to insert the horizontal bar. If your system does not yet support `\botsep`, you may have to be more clever about when to put in the bar and when to delete it. The author's previous method was to insert a bar if this was the first footnote on a page. The output routine would reset the first-footnote-on-the-page switch. However, this would occasionally fail in that the first footnote on the page would actually be generated before the output routine was called (e.g., if the paragraph is split on two pages). Then the `\firstfootnote` macro would be used which would hack the switches appropriately so that the next footnote would not get a bar. This involved setting a switch that the output routine cleared so that the output routine would not clear a second switch that indicated that a bar had already been output for that page. What a mess! Anyway, get a new version of TeX.

The macro `\footnote` provides automatically numbered footnotes. The numbers are started at 1—the macro pre-increments it.

There are three macros of characters for using for footnotes. These are `\upstar`, `\dagger`, and `\ddagger`.

## 2.9 Paragraphs

This section consists of macros for various hanging paragraphs. The `\hangbox` macro creates a box of width based on argument 1 containing argument 2. The remaining lines of the paragraph will be indented the same width. For example, `\hangbox to 30pt {foo}bar etc.`, will produce a paragraph containing bar etc., indented to 30 points with the first 30 points of the first line containing foo. The macros `\levelone`, `\leveltwo`, and `\levelthree` generate such hanging boxed paragraphs to 20 points, 40 points, and 60 points, respectively. However, the contents of the boxes are left justified in a twenty-point box that is right justified in the 40- or 60-point box. The following are uses of `\levelone`, `\leveltwo`, and `\levelthree`:

1. This is a short box followed by a long paragraph. Isn't it amazing to see what drivel can be published in the guise of an example. Put your ad here; to find out whether you can call 936-1212.
2. This is a medium box followed by a long paragraph. Isn't it amazing to see what drivel can be published in the guise of an example. For a good time call 767-8989.
3. This is a long box followed by a long paragraph. Isn't it amazing to see what drivel can be published in the guise of an example. For example, did you know that when the author finishes his Ph.D., he'll be looking for a teaching job? A reference to this article will pad his C.V.

The macros `\number` and `\nnumber` create indented paragraphs with boxes of 20 and 50 points respectively. That's right, `\number` is just like `\levelone`.

The `\indpar` macro takes the argument and creates an paragraph indented on both sides to 40 points. Normal paragraph indentation or paragraph separation must be done by you. A `\strut` has been inserted to get the correct line spacing between the paragraph and preceding and following text to handle risers and descenders properly. However, no `\parskip` glue is inserted and 1 point is inserted for the assumed `\lineskip`.

The `\hdr` macro creates a centered boldface heading consisting of its argument that makes a good section heading if you are not using book format.

## 2.10 List Definitions

Now that you know all about the paragraph macros, you might expect macros for doing numbered lists automatically. There are three levels of numbering. The first level uses `\list` followed by an argument which is the initial number for counting. Then `\item` is used to precede each item. You may use `\itemindent` to indent the same amount as `\item` for continuing the following paragraph, for example. The macro `\bitem` gives a centered bullet in a 20 point box starting the hanging paragraph.

The second level of counting is in roman numerals. Put the number you want to start counting from after the `\sublist`. Note that this number should be positive, so `-3` gives "iii." As you might expect, there are `\subitem` and `\subitemindent`.

The third level of counting is letters. Put the letter you want to start counting from after the `\subsublist`. And there are `\subsubitem` and `\subsubitemindent`.

## 2.11 Underlining and Boxes

This section consists of macros for doing various kinds of under- and overlining as well as lined boxes. The `\undertext` and `\overtext` macros underline and overline in horizontal mode. And `\leaderline` gives a leader of a rule.

The `\boxit` macro is from exercise 21.3 of the `TEX` manual. However, `\sizeboxit` makes the box a specific size. Also, `\boxitnoglue` boxes the box without 3 points of space on all sides. If you want to put `\boxit`'s in a `\valign` or put straight text inside, use `\Boxit` or `\Boxitnoglue`, which reverse horizontal and vertical mode. To put corner L's around a box, use `\Lboxit`.

To demonstrate interactive system output, it is useful to display the user entered data underlined. To underline the second half of a line, say `\type prompt>underlined text`. Use `\ttype` the same way for indented dialogue.

## 2.12 Penalties

Aren't these obvious. They do save space in macros over their expansions.

## 2.13 `\nofill \endnofill`

This is the first of the verbatim mode set. To use it, say `\nofill` followed by the text, followed by `\endnofill`. Line breaks appear exactly where they do in the input text. Exactly as many spaces appear in the output as in the input. The code is listed verbatim without page breaks. To allow page breaks, say `\allowbreak`. A blank line is generated if there is no page break. To have no space generated if there is no page break, use `\allowbreaknoglue`.

Tabs are not allowed in verbatim mode. This is because it is not clear how many spaces to generate for a tab. If you think you know better, say `\def\tab{definition}`.

## 2.14 `\startcode \endcode`

The `\startcode` `\endcode` sequence produces verbatim code in `\displayfont`. See the previous section and the code for more details.

In `\startcode` mode, `\le` and `\ge` produce  $\leq$  and  $\geq$ , respectively. These revert back to their former meanings at the end.

## 2.15 Verbatim Mode Using `$$\halign$$`

Verbatim mode is just like `\startcode` mode except that the calling conventions are different and it may appear in an `\halign`. To use it precede the code with `\halign{` and follow the code with a right brace on its own line. To allow a page break, code `\breakhere%`. Note that the `%` is required.

The `\threecol` macro generates `\verbatim` mode except with three columns instead of one. To put a box around verbatim code, say `\Boxit{\verbatim{code}}`.

## 2.16 Notes

Notes are useful to provide descriptions of things that you want to fix. The description of the file `MNOTES.TEX` appears later.

## 2.17 Index Macros

An index package is described in Vol. 1, No. 1 of TUGboat.

## 2.18 Defer Mode

Defer mode is useful for specifying an entire page that is to appear as soon as possible. The description of the file `DEFER.TEX` appears later.

## 2.19 Table of Contents

This set of macros generates a table of contents compatible with book format. The description of the file `MTOFC.TEX` appears later.

## 2.20 Interesting Hacks

This section consists of interesting macro hacks that are useful for one and all.

The `\ifnull` macro determines if argument 1 is null. If so it expands argument 2; otherwise, argument 3 is expanded. To call say: `\ifnulltext\then{code}\else{code}`.

The `\bracex`, `\dnbrace`, and `\upbrace` macros are from page 103 of the TeX manual. On the other hand, `\blackslug` is from page 167.

The `\boxtop` macro sets the baseline at the top of the box. This is useful for lining up variable sized boxes at the top. For example, to line up `\hbox` par's in a `\halign`, use `\boxtop` around the `\hbox` par's.

The `\topspace` and `\magnify` macros is from the new version of `BASIC.TEX`.

There are two macros for playing with counters. Use `\setq` to set the control sequence which is the first argument to the counter number in the second argument. Use `\advcountq` to increment the counter in the control sequence. With these macros, you can save counters for what they are really needed for: setting up the correct numbers for output routines.

The `\done` macro goes at the end of the document.

The `\capitalpar` macro creates a paragraph like the "Gentle reader" at the start of the TeX manual.

## 2.21 Default Options

See `\startcode` and `\endcode` for a description of `\displayfont`. See section 1.6 for a description of page numbering.

## 2.22 Documentation

Every macro package should have some.

## 3 MBOOK.TEX

The book format macro package is the most developed of the macro formats. The `\bookoutput` routine handles `\titlepage` (it sets `\tpage` to T), as well as proper placement of the page numbers. The page

numbers will appear on the top of the page if `\pagenumberarea` is T, on the bottom if B; otherwise, no page numbers appear. It defaults to T.

The `\pagenumberregion` macro defines the format of the page heading (or footing). It defaults to the macro `\boxpagenumberregion`, which produces the format in the TeX manual and this document. The right and left headings are specified by `\titlemark{right}{left}`.

To get a page without a page heading, say `\titlepage`.

Chapters, sections, subsections, and diminished sections are all numbered automatically. To use, say `\chapterbegin`, `\sectionbegin`, `\subsectionbegin`, or `\dimsectionbegin`, all followed by the chapter or section name in braces. For an unnumbered chapter, such as an appendix or a table of contents, use `\specialbegin` followed by the name in braces. Use `\settitle` to set the left and right headings if you aren't using the other macros in this paragraph.

To get data for a table of contents, use `\inittofc{filename}`.

#### 4 MACACM.TEX

This produces 25% oversized output for the ACM camera-ready copy specifications. At the start of the paper, code `\useacmformat`. Then define `\title` and `\authors`. Then define the title portion of the paper. Next, say `\endoftitle`. When done with the paper, say `\endofpaper\end`.

Note that `\defer` does not work with this format.

#### 5 OPLAIN.TEX

This produces output unadorned with page numbers or anything.

#### 6 OBASIC.TEX

This produces output with page numbers as in BASIC.TEX.

#### 7 OWOODS.TEX

This produces output with page numbers on the bottom of the page with hyphens around the numbers as popularised by Don Woods at Stanford.

#### 8 SBLOCK.TEX

This produces block style paragraphs with about 6 points of space between paragraphs.

#### 9 SBASIC.TEX

This produces indented paragraphs as in BASIC.TEX.

#### 10 MNOTES.TEX

This file contains macros to generate notes to the writer. Say `\initnotes` to create the file. Say `\sendnotes{text}` to output text. The `\putnotes` macro takes the notes and outputs them in the listing.

#### 11 DEFER.TEX

Defer mode is used to produce a floating figure that takes one or more whole pages. Like `\topinsert` for floating figures, it is used in vertical mode. However, defer mode handles multiple page figures and will as keeping several figures in the order specified.

To use, say `\defer` followed by the figure followed by `\enddefer`.

Defer mode does not work with multicolumn formats.

## 12 MTOFC.TEX

The table-of-contents package produces a table of contents based on the data files produced if `\inittofc` is used in book format. To use, say `\begintofc` followed by `\chaptertofc`, `\sectiontofc`, etc., macros each followed by a title in braces and a page number terminated by a period. Use `\endtofc` at the end of the table of contents.

## 13 Acknowledgments

This paper could not have been written and published without TeX designed by Donald E. Knuth. He also provided some advice and encouragement, as did Luis Trabb-Pardo. Many of the macros were written with the help of Jim Boyce. Brent Hailpern designed the original defer mode. Max Diaz provided additional suggestions and several macros as well. Denny Brown suggested some of the macros and the idea of supporting many formats. My advisor, Gio Wiederhold, gave encouragement and enlightened criticism through his attempts to use these macros. The staff of the American Mathematical Society were very helpful and interested in getting this manuscript written and published.

## 14 The Macros

The source for the macro package follows.

## Standard BASIC Stuff

```

\chcode '45+5      % % **** N.B. this must be first
\chcode '173+1     % {
\chcode '176+2     % }
\chcode '44+3      % $
\chcode '26+4      % @
\chcode '43+6      % #
\chcode '136+7     % †
\chcode '1+8       % Δ

```

```

% Shorthands for certain definitions
\def \trace{\chpar0+} \trace'1400345
\def \jpar{\chpar1+}
\def \hpen{\chpar2+}
\def \ragged{\chpar8+}

```

```

% centerings
\def \lft#1{\hfill #1\hfill }
\def \ctr#1{\hfill #1\hfill }
\def \rt#1{\hfill #1}
\def \top#1{\vfill #1\vfill }
\def \mid#1{\vfill #1\vfill }
\def \bot#1{\vfill #1}

```

```

\def \ljustline#1{\hbox to size{#1\hss}}
\def \ctrline#1{\hbox to size{\hss #1\hss}}
\def \rjustline#1{\hbox to size{\hss #1}}

```



```

\def \ldots{(. \condthinspace. \condthinspace.)}
\def \cdots{\char '401\condthinspace\char '401\condthinspace\char '401}
\def \ldotss{(. \condthinspace. \condthinspace. \condthinspace.)}
\def \cdotss{\cdots \condthinspace}
\def \ldotssm{(. \condthinspace. \condthinspace. \condthinspace. \condthinspace.)}
\def \vdots{\vbox{\baselineskip 4pt \vskip 6pt \hbox{.}\hbox{.}\hbox{.}}}

\def \cpile #1{\vcenter {\halign {\hfill ## $\hfill \cr #1}}}
\def \lpile #1{\vcenter {\halign {$## $\hfill \cr #1}}}
\def \rpile #1{\vcenter {\halign {\hfill ## $\cr #1}}}

\def \null{\hbox {}}.

\def \spose #1{\hbox to 0pt{#1\hss}}

\def \log{\mathop{\char l\char o\char g}\limitswitch}
\def \lg{\mathop{\char l\char g}\limitswitch}
\def \ln{\mathop{\char l\char n}\limitswitch}
\def \lim{\mathop{\char l\char i\char m}}
\def \limsup{\mathop{\char l\char i\char m\char s\char u\char p}}
\def \liminf{\mathop{\char l\char i\char m\char i\char n\char f}}
\def \sin{\mathop{\char s\char i\char n}\limitswitch}
\def \cos{\mathop{\char c\char o\char s}\limitswitch}
\def \tan{\mathop{\char t\char a\char n}\limitswitch}
\def \cot{\mathop{\char c\char o\char t}\limitswitch}
\def \sec{\mathop{\char s\char e\char c}\limitswitch}
\def \csc{\mathop{\char c\char s\char c}\limitswitch}
\def \max{\mathop{\char m\char a\char x}}
\def \min{\mathop{\char m\char i\char n}}
\def \sup{\mathop{\char s\char u\char p}}
\def \inf{\mathop{\char i\char n\char f}}
\def \det{\mathop{\char d\char e\char t}}
\def \exp{\mathop{\char e\char x\char p}\limitswitch}
\def \Pr{\mathop{\char P\char r}}
\def \gcd{\mathop{\char g\char c\char d}}
\def \lcm{\mathop{\char l\char c\char m}}
\def \choose{\comb()}
\def \leftset{\mathopen{\{(.)}}
\def \rightset{\mathclose{\{.}}}}
\def \modop{<\, \mathbin{\char m\char o\char d}\penalty 900<\,}
\def \mod#i{\penalty 0; (\char m\char o\char d\, \, #1)}
\def \eqv{\mathrel{\char'421 }}
\def \neqv{\mathrel{\not\eqv}}

```

```

\def\eqalign#1{\baselineskip15pt\lineskip3pt
  \vcenter{\halign{\hfill$\dispstyle{##}$@$\dispstyle{\null##}$\hfill
    \cr#1}}}
\def\eqalignno#1{\baselineskip15pt\lineskip3pt
  \vbox{\tabskip 0pt plus 1000pt minus 1000pt
    \halign to size{\hfill$\dispstyle{##}$\tabskip 0pt
      @$\dispstyle{\null##}$\hfill\tabskip 0pt plus 1000pt minus 1000pt
      @\hfill$ ##$\tabskip 0pt\cr#1}}}
\def\twoline#1#2#3{\vbox{\hbox to size{$\quad\dispstyle{#1}$\hfill}
  \vskip#2\hbox to size{\hfill$\dispstyle{#3}\quad$}}}
\def\chop to#1pt#2{\hbox{\lower#1pt\hbox{\lower100pt\hbox{\raise100pt
  \hbox{$\dispstyle{#2}$}}}\vskip-100pt}} % pretends that #2 is #1pt deep

```

### • Font Definitions and Related Macros

```

\chcode'100+12 % allow @ on this page to be parsed correctly

% font definitions for 8, 9, and 10 point fonts and friends
\font @+cmathx
\font a+cmr10 \font b+cmr9 \font c+cmr8 \font d+cmr7 \font e+cmr6 \font f+cmr5
\font g+cmi10 \font h+cmi9 \font i+cmi8 \font j+cmi7 \font k+cmi6 \font l+cmi5
\font G+cmti10 \font H+cmti9 \font I+cmti8
\font m+cmsc10
\font n+cms10 \font o+cms9 \font p+cms8
\font q+cmb10 \font r+cmb9 \font s+cmb8
\font t+cmtt \font T+cmtt9 \font U+cmtt8
\font u+cmsy10 \font v+cmsy9 \font w+cmsy8 \font x+cmsy7 \font y+cmsy6 \font z+cmsy5

% font definitions for random desired fonts
\font ;+cmtitl
\font <+cmssb \font =+cmss12 \font >+cmss8 \font ?+cmsss8

% font request macros
\let \usefont=:
\def \curfont #1{\usefont #1\def\fontcode{#1}}
\let \:=\curfont

```

```

% font name macros
\def \loadfont#1#2#3{\font #1+#2 \gdef #3{\: #1}#3}
\def \big{\loadfont D{cmr12}{\big}}
\def \ms25{\loadfont A{ms25}{\ms25}}
\def \nons{\loadfont B{nons}{\nons}}
\def \peni11{\loadfont P{peni11}{\peni11}}
\def \stan70{\loadfont S{stan70}{\stan70}}
\def \biggfont{\loadfont C{cmr10 at 20pt}{\biggfont}}
\def \bigggfont{\loadfont E{cmr10 at 30pt}{\bigggfont}}
\def \cmrten{\:a}
\def \cmrnine{\:b}
\def \cmreight{\:c}
\def \cmrseven{\:d}
\def \cmrsix{\:e}
\def \cmrfive{\:f}
\def \cmiseven{\:j}
\def \cmisix{\:k}
\def \cmifive{\:l}
\def \cmscten{\:m}
\def \cmtitl{\:;}
\def \cmssb{\:<}
\def \cmss12{\:=}
\def \cmsseight{\:>}
\def \cmsseight{\:??}

% font family definitions
\def \tenpoint{\baselineskip 12pt
  \dispskip 12pt plus 3pt minus 9pt
  \dispaskip 0pt plus 3pt
  \dispbskip 7pt plus 3pt minus 4pt
  \def \strut{\lower 3.5pt
    \vbox to 12pt{}}% i.e., \lower 1pt+.25em\vbox to 2pt+1em{}
  \def \rm{\:a}
  \def \sl{\:n}
  \def \bf{\:q}
  \def \it{\:G}
  \def \mi{\:g}
  \def \tt{\:t}
  \def \sy{\:u}
  \def \sc{\:m}
  \def \biglp{\mathopen {\vcenter {\hbox {\:0\char '}}}}
  \def \bigrp{\mathclose{\vcenter {\hbox {\:0\char '1}}}}
  \def \9{\hskip 5pt}
  \mathrm adf
  \mathit gjl
  \mathsy urz
  \rm \usertenpoint}
\def \usertenpoint{}
\def \usetenpoint{\gdef\fontsize{\tenpoint}\tenpoint}

```

```

\def \ninepoint{\baselineskip 11pt
  \dispskip 11pt plus 3pt minus 8pt
  \dispaskip 0pt plus 3pt
  \dispbskip 6pt plus 3pt minus 3pt
  \def \strut{\lower 3.25pt\vbox to 11pt{}}% see tenpoint for explanation
  \def \rm{\:b}
  \def \sl{\:o}
  \def \bf{\:r}
  \def \it{\:H}
  \def \mi{\:h}
  \def \tt{\:T}
  \def \sy{\:v}
  \def \biglp{\mathopen {\hbox{\:a{}}}}
  \def \bigrp{\mathclose{\hbox{\:a{}}}}
  \def \9{\hskip 4.625pt}
  \mathrm bef
  \mathit hkl
  \mathsy vyz
  \rm \userninepoint}
\def \userninepoint{}
\def \useninepoint{\gdef\fontsize{\ninepoint}\ninepoint}

\def \eightpoint{\baselineskip 9.5pt
  \dispskip 5pt plus 3pt minus 2pt
  \dispaskip 0pt plus 3pt
  \dispbskip 5pt plus 3pt minus 2pt
  \def \strut{\lower 2.75pt\vbox to 9.5pt{}}% see tenpoint for explanation
  \def \rm{\:c}
  \def \sl{\:p}
  \def \bf{\:s}
  \def \it{\:I}
  \def \mi{\:i}
  \def \tt{\:U}
  \def \sy{\:w}
  \def \biglp{\mathopen {\hbox {\:a{}}}}
  \def \bigrp{\mathclose{\hbox {\:a{}}}}
  \def \9{\hskip 4.25pt}
  \mathrm cef
  \mathit ikl
  \mathsy wyz
  \rm \usereightpoint}
\def \usereightpoint{}
\def \useeightpoint{\gdef\fontsize{\eightpoint}\eightpoint}

\mathex 0

% definitions of large parentheses
\def \biggpl{\mathopen{\vcenter{\hbox{\:0\char'22}}}}
\def \biggrp{\mathclose{\vcenter{\hbox{\:0\char'23}}}}
\def \bigggpl{\mathopen{\vcenter{\hbox{\:0\char'40}}}}
\def \bigggp{\mathclose{\vcenter{\hbox{\:0\char'41}}}}

```

```
% definitions of glue
\def \qqquad{\quad\quad}
\def \xskip{\hskip 7pt plus 3pt minus 4pt}
\def \yskip{\penalty-50\vskip 3pt plus 3pt minus 2pt}
\def \yyskip{\goodbreak\vskip 6pt plus 6pt minus 4pt}
```

---

#### • Definitions of Odd Characters

```
\chcode'272+'3072 % this makes formulas like "$x:=x+1$" and "$f\?:X\to Y$" work
```

```
% ttchar puts the char into a \tt fixed width box
```

```
\def\ttchar#1{\save1\hbox{\ } \hbox to 1wd1{\hskip0pt plus1000pt minus1000pt
#1\hskip0pt plus1000pt minus1000pt}}
```

```
% fontclassify selects the right char based on what the current font is
```

```
\def\fontclassify#1#2#3{\ifmode{#1}
  \else{\if t\fontcode{#2}
  \else{\if T\fontcode{#2}
  \else{\if U\fontcode{#2}
  \else{(#3)}}}}}
```

```
%#1 is math, #2 is tt, #3 is others
```

```
% char macro definitions
```

```
\def\down{\fontclassify{\mathrel{\char'443}}{\ttchar{\sy\char'43}}{\sy\char'43}}
```

```

\def\alpha{\fontclassify{\char'213}{\ttchar{\mi\char'13}}{\mi\char'13}}
\chcode'2+13
\def\beta{\fontclassify{\char'214}{\ttchar{\mi\char'14}}{\mi\char'14}}
\chcode'3+13
\def\^{\fontclassify{\mathbin{\char'536}}{\ttchar{\if t\fontcode{\:z\char'136}
\else{\if T\fontcode{\:z\char'136}\else{\:z\char'136}}}{\sy\char'136}}
\chcode'4+13
\def\~{\fontclassify{\char'472}{\ttchar{\sy\char'72}}{\sy\char'72}}
\chcode'5+13
\def\epsilon{\fontclassify{\char'217}{\ttchar{\mi\char'17}}{\mi\char'17}}
\chcode'6+13
\def\pi{\fontclassify{\char'231}{\ttchar{\mi\char'31}}{\mi\char'31}}
\chcode'7+13
\def\lambda{\fontclassify{\char'225}{\ttchar{\mi\char'25}}{\mi\char'25}}
\chcode'10+13
\def\omega{\fontclassify{\char'461}{\char'25}{\sy\char'61}}
\chcode'16+13
\def\theta{\fontclassify{\char'245}{\ttchar{\mi\char'45}}{\mi\char'45}}
\chcode'17+13
\def\llcorner{\fontclassify{\mathrel{\char'432}}{\ttchar{\sy\char'32}}{\sy\char'32}}
\chcode'20+13
\def\lrcorner{\fontclassify{\mathrel{\char'433}}{\ttchar{\sy\char'33}}{\sy\char'33}}
\chcode'21+13
\def\lshch{\fontclassify{\mathbin{\char'534}}{\ttchar{\sy\char'134}}{\sy\char'134}}
\chcode'22+13
\def\lll{\fontclassify{\mathbin{\char'533}}{\ttchar{\sy\char'133}}{\sy\char'133}}
\chcode'23+13
\def\llv{\fontclassify{\char'470}{\ttchar{\sy\char'70}}{\sy\char'70}}
\chcode'24+13
\def\exists{\fontclassify{\char'471}{\ttchar{\sy\char'71}}{\sy\char'71}}
\chcode'25+13
\def\@{\fontclassify{\mathbin{\char'412}}{\char'26}{\sy\char'12}}

\def\w{\fontclassify{\mathrel{\char'444}}{\ttchar{\sy\char'44}}{\sy\char'44}}
\chcode'27+13
\def\_{\fontclassify{\char'465}{\char'32}{\sy\char'65}}
\chcode'30+13
\def\rarrow{\fontclassify{\mathrel{\char'441}}{\ttchar{\sy\char'41}}{\sy\char'41}}

\def\lshch{\fontclassify{\mathrel{\char'430}}{\ttchar{\sy\char'30}}{\sy\char'30}}
\chcode'32+13
\def\lshch{\fontclassify{\mathrel{\char'434}}{\ttchar{\sy\char'34}}{\sy\char'34}}
\chcode'33+13
\def\le{\fontclassify{\mathrel{\char'424}}{\hbox{\sponse{\char'32}<}}{\sy\char'24}}

\def\ge{\fontclassify{\mathrel{\char'425}}{\hbox{\sponse{\char'32}>}}{\sy\char'25}}

\def\equiv{\fontclassify{\mathrel{\char'421}}{\ttchar{\sy\char'21}}{\sy\char'21}}
\chcode'36+13
\def\ve{\fontclassify{\mathbin{\char'537}}{\ttchar{\:z\char'137}}{\sy\char'137}}
\chcode'37+13
\def\#\{\fontclassify{\char'561}{\char'43}{\sy\char'161}}

\def\$\{\fontclassify{\char'577}{\char'44}{\sy\char'177}}

```

```

\def\%{\char'45}

\def\@{\fontclassify{\char'574}{\char'100}{\sy\char'174}}
\chcode'100+13
\def\|\{\fontclassify{\mathbin{\char'404}}{\char'134}{\sy\char'404}}

\def\up{\fontclassify{\mathrel{\char'442}}{\char'136}{\sy\char'42}}

\def\larrow{\fontclassify{\mathrel{\char'440}}{\char'137}{\sy\char'40}}

\def\lbrace{\fontclassify{\mathopen{\char'546610}}{\char'173}{\sy\char'146}}

\def\orbar{\fontclassify{\char'552614}{\char'174}{\sy\char'152}}

\def\rbrace{\fontclassify{\mathclose{\char'547611}}{\char'176}{\sy\char'147}}

\def \uparrow{\up}

\def \sharp{\#}

\def \seal{\stan70 S}

\let \space=\ % for defining \ to be \hbox{\space} in \tt

\def\sp{\tt\char'40}

```

---

#### Redefinitions of One Character Macros

```

\let \space=\ % for defining \ to be \hbox{\space} in \tt

\let \Mthinspace=\.
\let \Mopospace=\>
\let \Mthickspace=\;
\let \Mcondthinspace=\>
\let \Mnegthinspace=\!
\let \Mignorespace=\!
\let \Mnegthickspace=\?
\let \Mnegospace=\<
\let \Mnegcondthinspace=\<

% new long names work anywhere
\def \thinspace{\ifmode{\Mthinspace}\else{\Mthinspace}}
\def \opospace{\ifmode{\Mopospace}\else{\Mopospace}}
\def \thickspace{\ifmode{\Mthickspace}\else{\Mthickspace}}
\def \condthinspace{\ifmode{\Mcondthinspace}\else{\Mcondthinspace}}
\let \negthinspace=\Mnegthinspace
\def \negthickspace{\ifmode{\Mnegthickspace}\else{\Mnegthickspace}}
\def \negospace{\ifmode{\Mnegospace}\else{\Mnegospace}}
\def \negcondthinspace{\ifmode{\Mnegcondthinspace}\else{\Mnegcondthinspace}}

```

```
% redefine old names to match new names
\let \,=\thinspace
\let \>=\opSPACE
\let \;=\thickSPACE
\let \_>=\condthinspace
\let \?=\negthickSPACE
\let \<=\negopSPACE
\let \_<=\negcondthinspace
```

---

• **Make Some Math Things Work Anywhere**

```
% save old definitions
\let \Msection=\section
\let \Mdag=\dag
\let \Mddag=\ddag
\let \MP=\P
\let \Mcopyright=\copyright
\let \Msterling=\sterling
\let \Mbullet=\bullet
\let \Mcirc=\circ

% let these work in any mode using old math mode definitions
\def \section{\ifmode{\Msection}\else{\Msection$}}
\def \dag{\ifmode{\Mdag}\else{\Mdag$}}
\def \ddag{\ifmode{\Mddag}\else{\Mddag$}}
\def \P{\ifmode{\MP}\else{\MP$}}
\def \copyright{\ifmode{\Mcopyright}\else{\Mcopyright$}}
\def \sterling{\ifmode{\Msterling}\else{\Msterling$}}
\def \bullet{\ifmode{\Mbullet}\else{\Mbullet$}}
\def \circ{\ifmode{\Mcirc}\else{\Mcirc$}}
% Note that \$ is defined with the odd characters and @ now does the right
% thing in any mode as does \@
```

---

• **Page Numbering**

```
% uses two flags:
%   \indefer mode is T when in defer mode
%   \deferredpage is T when there is a piece of a page being deferred

\def\advpagecount{\if T\indefer mode{\advpagecountone \setcount0\highestpagenumber}
\else{\if T\deferredpage{\setcount0\savedpagecount
\gdef\deferredpage{F}}
\else{\advpagecountone \setcount0\highestpagenumber}
}
}
\def\deferredpage{F}
\def\indefer mode{F}

\def\incpagecount{\gdef\advpagecountone{\advcountq{\highestpagenumber}}}
\def\decpagecount{\gdef\advpagecountone{\setcount9\highestpagenumber
\advcount9by-1
\setq{\highestpagenumber}9}}
```



```

\def\setpagecount#1{\setcount9 #1
  \ifpos9{\incpagecount\advcount9 by -2}
  \else{\decpagecount\advcount9 by 2}
  \setq{\highestpagenumber}9
}

```

---

• **“output, Style, Format Routines**

```

\def\normal{\resetsize \fontsize \parstyle}

\def\resetsize{\normalhsize \normalvsize}

\def\everyoutput{} % this is something that is in every output routine

% start of format descriptions
\def\usebookformat{\input mbook }

\def\usebasicformat{\usebasicstyle \usebasicoutput }

% ACM oversize format for Versatec (camera ready copy)
\def\useacmformat{\input macacm }
% To use, code \useacmformat at the start of the paper.
% Then define \title and \authors
% then define the title portion, followed by \endoftitle
% when you are all done \endofpaper\end

\def\useplainoutput{\input oplain }

\def\usebasicoutput{\input obasic }

\def\useWoodsoutput{\input owoods }

\def\useblockstyle{\input sblock }

\def\usebasicstyle{\input sbasic }

% look at \useplainoutput and \useblockstyle for the minimum needed
% in output and style routines
% Format routines are simply output and style together. Note that
% other related macros and definitions may be included also.

```

---

• **Footnotes**

```

% normal footnote
\def\footnote#1#2{#1\botinsert{\eightpoint\hbox par size{#1#2}}}

% numbered footnote
\def\nfootnote#1{\advcountq{\footnotenumbe}!\
  $†{\footnotenumbe}$!\
  \botinsert{\eightpoint\hbox par size{†{\footnotenumbe}$#1}}}
\def\footnotenumbe{0}

```

```
\botsep{\vskip15pt \hrule width5pc\vskip 3pt}
```

```
% footnote mark characters
```

```
\def\upstar{\lower 3pt \hbox{${\hbox{*}}$}}
```

```
\def\dagger{\lower 2pt \hbox{${\Mdag}$}}
```

```
\def\ddagger{\lower 2pt \hbox{${\Mddag}$}}
```

### • Paragraphs

```
\def\hangbox to #1 #2{\par\hangindent #1\noindent
  \hbox to #1{#2}\!}
```

```
\def\levelone#1{\hangbox to 20pt {#1\hfill}}
```

```
\def\leveltwo#1{\hangbox to 40pt {\hbox to 20pt{\hfill}}#1\hfill}}
```

```
\def\levelthree#1{\hangbox to 60pt {\hbox to 40pt{\hfill}}#1\hfill}}
```

```
\def\number#1{\levelone{#1}}
```

```
\def\nnumber#1{\hangbox to 50pt {#1\hfill}}
```

```
\def\indpar#1{\par
```

```
  \save9\hbox to size{}
```

```
  \save9\hbox{\box9\hskip-40pt} % width minus 40pt
```

```
  \hsize 1wd9
```

```
  \vskip1pt
```

```
  \leveltwo{ }\strut#1\strut}\par\normalhsize
```

```
  \vskip1pt}
```

```
\def \hdr#1{\par\goodbreak\yyskip\ctrline{\bf #1}\posthdrskip}
```

```
\def \posthdrskip{\par\badbreak\vskip 5pt\badbreak}
```

```
\def \sectionskip{\par\excellentbreak\vskip 24pt plus 12pt minus 6pt}
```

### • List Definitions

```
\def \list#1{\xdef{\listcounter{#1}}}
```

```
\def \item{\advcountq{\listcounter}
```

```
  \levelone{\listcounter.}}
```

```
\def \itemindent{\levelone{}}
```

```
\def \bitem{\levelone{\hfill\bullet}} % this centers the bullet. see \levelone
```

```
\def \sublist#1{\xdef{\sublistcounter{#1}}}
```

```
\def \subitem{\advcountq{\sublistcounter} % leaves count in \count9
```

```
  \setcount9 -\sublistcounter % we want roman numerals
```

```
  \leveltwo{\count9.}}
```

```
\def \subitemindent{\leveltwo{}}
```

```
\def \subsublist#1{\xdef{\subsublistcounter{#1}} % should be a letter
```

```
\def \subsubitem{\advcountq{\subsublistcounter}
  \levelthree{\char\subsublistcounter.}}
```

```
\def \subsubitemindent{\levelthree{}}
```

---

### • Underlining and Boxes

```
\def \undertext #1{\underline{\hbox{#1}}}$} % underline in horizontal mode
```

```
\def \overtext #1{\overline{\hbox{#1}}}$} % overline in horizontal mode
```

```
\def \leaderline{\leaders\hrule\hfill}
```

```
\def \boxit#1{\vbox{\hrule\hbox{\vrule\hskip3pt
  \vbox{\vskip3pt#1\vskip3pt}\hskip3pt\vrule}\hrule}}
```

```
\def \sizeboxit to#1by#2 #3{\vbox{\hrule\hbox to #1{\rule\hss
  \vbox to #2{\vss#3\vss}\hss\vrule}\hrule}}
```

```
\def \boxitnoglue#1{\vbox{\hrule\hbox{\vrule
  \vbox{#1}\vrule}\hrule}}
```

% Boxit and Boxitnoglue are like boxit and boxitnoglue except that horizontal  
% and vertical modes are reversed.

```
\def \Boxit#1{\hbox{\vrule\vbox{\hrule\vskip3pt
  \hbox{\hskip3pt#1\hskip3pt}\vskip3pt\hrule}\vrule}}
```

```
\def \Boxitnoglue#1{\hbox{\vrule\vbox{\hrule
  \hbox{#1}\hrule}\vrule}}
```

% Lboxit puts L's around box instead of rules

```
\def \Lboxit to #1 by #2 #3{\def\hsplitrule{\hbox to #1{\vbox{\hrule width .25in}
  \hfill
  \vbox{\hrule width .25in}}}}
```

```
\def \vsplitrule{\vbox to #2{\hbox{\vrule height .25in}\vfill
  \hbox{\vrule height .25in}}}
```

```
\vbox{\lineskip Opt
  \baselineskip Opt
  \hsplitrule
  \vbox to #2{\hbox to #1{\vsplitrule
    \hfill
    \vbox to #2{\vfill#3\vfill}
    \hfill
    \vsplitrule}}
  \hsplitrule
}}
```

```
\def \type #1>#2{\par\indpar{\displayfont #1\under{#2}}} % type a line (as in dialogue)
% the second argument is underlined, good for prompts
```

```
\def \ttype #1>#2{\par\noindent{\displayfont#1\under{#2}}\par}
% type a line (as in dialogue)
```

---

 • Penalties

```

\def\badbreak{\penalty1000}

\def\goodbreak{\penalty-100}

\def\excellentbreak{\penalty-1000}

```

---

## • "nofill" "endnofill"

```

% To use, code:
% \nofill
% statements
% \endnofill
%
% The code is listed verbatim without any page breaks.
% To allow a page break, put \allowbreak on a line. If there
% is no break, a blank line is generated.
%
% Note that \fontsize must be defined to be your normal size of type, such
% as \tenpoint
%
% Font is not changed

% Use of tabs in verbatim mode will give an error message.

% Define \<cr> to be \CR when enabled
\chcode'15+12\def\
{\CR}\chcode'15+5 %

\def\nofill{\parskip Opt
\chcode'11+13           % define tab to give an error
\chcode'15+13           % define <return> to generate \cr
\chcode'40+13           % define space to generate \<space> (a real space)
\gdef\ {\hbox{\space}} % make space exactly one unshrinkable space
\gdef\CR{\par\badbreak\noindent\hbox{\!\,}}}}

\def\endnofill{\par\badbreak % force glue to this page
\vskip-11pt
\chcode"11+10           % define tab to be a space
\chcode'15+5           % define <return> be a end of line
\chcode'40+10           % define space to be a space
\let\ =\space          % make "\ " as normal
\normal}

\def\goodgele{\chcode'34+13 % ≤
\let \≤=\le
\chcode'35+13           % ≥
\let \≥=\ge
}

```

```

\def\normalgele{\chcode'34+12 % ≤
\let \le=\negcondthinspace
\chcode'35+12 % ≥
\let \ge=\condthinspace
}

```

---

• Verbatim Mode "startcode and "endcode

```

% To use, code:
% \startcode
% statements
% \endcode
%
% The code is listed verbatim without any page breaks.
% To allow a page break, put \allowbreak on a line. If there
% is no break, a blank line is generated.
%
% \startcode supplies 4 pt of glue
% \endcode supplies 5 pt of glue
% The code is printed in \displayfont mode
% To avoid glue, code \startcodenoglu or \endcodenoglu

% Note that \fontsize must be defined to be your normal size of type, such
% as \tenpoint

% Use of tabs in verbatim mode will give an error message.

% Define \<tab> to be \tab when enabled
\chcode'11+12\def\    {\tab}\chcode'11+10
% will cause an error message unless \tab is defined

\def\startcodenoglu{\par
\displayfont
\nofill
\goodgele
}

\def\endcodenoglu{\endnofill
\normalgele
\fontsize
}

\def\startcode{\par\excellentbreak\vskip 5pt plus 1pt minus 1pt\startcodenoglu}
\def\endcode{\endcodenoglu\excellentbreak\vskip 6pt plus 1pt minus 1pt}

\def\startoutput{\par\excellentbreak\vskip 5pt plus 1pt minus 1pt{\tenpoint
$\down\qqad\down\qqad\down\qqad\down\qqad\down\qqad\down\qqad\down$\par}
\vskip 6pt plus 1pt minus 1pt}

\def\allowbreaknoglu{\par\badbreak\vskip-11pt\excellentbreak}

\def\allowbreak{\allowbreaknoglu\vskip 11pt plus 1pt}

```

---

**•• Verbatim Mode Using `ftft` "halignftft"**

```

% To use, code the following:
% \verbatim{
% follow with code
% } terminates verbatim mode.
% Note that \verbatim stuff will not be broken across page boundaries.

% To allow a break, use \noalign{\excellentbreak}%
% or \breakhere%
% Note the absence of spaces in the above.
% Note that the % is necessary to avoid an extra line generated.

% Note that \fontsize must be defined to be your normal size of type, such
% as \tenpoint

% These macros rely upon the definitions of \<cr> and \<tab> on the previous page.
% Use of tabs in verbatim mode will give an error message.

\def\verbatim{\nofill
\gdef\CR{\cr\noalign{\badbreak}}
\goodgele
\verbatimgenerate}

\def\verbatimgenerate#1{{\displayfont$$\halign to size{##\hfill\cr#1}$$}
\endnofill
\normalgele
}

\def\breakhere{\noalign{\excellentbreak\vskip 11 pt}}

\def\threecol{\nofill
\gdef\CR{\cr\noalign{\badbreak}}
\goodgele
\threecolgenerate}

\def\threecolgenerate#1{{\displayfont\halign{##\hfill@##\hfill@##\hfill\cr#1}}%
\endnofill
\normalgele
}

```

---

**• Notes**

```

% \sendnotes creates a list of entries which will be output when
% \putnotes is used. This should be at the end of the manuscript.
% use \initnotes to initialize notes

\def \initnotes{\input mnotes }

```

• **Index Macros**

```
\def \initindex{\input mindex }
% see TUGboat (Vol. 1, No. 1) for an index package.
```

• **Defer Mode**

```
\def\defer{\input defer }
```

• **Table of Contents**

```
\def\beginofc{\input mtofc }
```

• **Interesting Hacks**

```
\def\ifnull#1\then#2\else#3{\def\jnk{#1?}\if?#1{#2}\else{#3}}
% to use \ifnull #1\then<true clause>\else<false clause>

\def\bracex{\leaders\hrule height 1.5pt \hfill}
\def\dnbrace{${\char'772$\bracex${\char'775
  \char'774$\bracex${\char'773$}
\def\upbrace{${\char'774$\bracex${\char'773
  \char'772$\bracex${\char'775$}

\def \TEX{\hbox{\rm T\hskip-.1667em\lower.424ex\hbox{E}\hskip-.125em X}}

\def\blackslug{\hbox{\hskip 1pt \vrule width 4pt height 6pt depth 1.5pt
  \hskip 1pt}}

\def\boxtop#1{\save9#1\lower 1ht9\box9}

\def\topspace{(\hrule height0pt)\vskip}
  % e.g. "\topspace 1in" puts an inch of space at the top of a page

\def\setq#1#2{\ifpos#2{\gdef#1{}}
  \else{\gdef#1{-} \setcount#2 -\count#2}
  \xdef#1{#1\count#2}
  \setcount#2#1} % notice how we restore \count#2

\def\advcountq#1{\setcount9#1
  \advcount9by1
  \setq{#1}9}

\def\magnify#1{\chpar12=#1} % operand is magnification times 1000

\def\done{\par\vfill\end}
```

```
% To put a big capital letter begining a paragraph; #1 = indent for (2 or 3)
% lines, #2 = letter, #3 = paragraph
\def\capitalpar#1#2#3{\save9\hbox par size{\ragged 1000000
  \if2#1{{1 \linebreak 2}} % find out how much
  \else{{1 \linebreak 2 \linebreak 3}}} % to move up
  \vbox{\hbox{\biggfont #2}
  \vskip -1ht9
  \save9\hbox{\biggfont #2}
  \hbox par size{\hangindent 1.3wd9 for #1{ }#3}}
}
```

```
\def\ie{{\sl i.e.}}
\def\eg{{\sl e.g.}}
```

---

#### • Default Options

```
\def\displayfont{\ninepoint\tt}
\setpagecount{1}
```

---

#### • Documentation of Use of Counters and Boxes

```
% Counters and use
% 0 the page number to appear on current page. Valid only in \output,\send,\mark
% 1 unused
% 2 unused
% 3 unused
% 4 unused
% 5 unused
% 6 unused
% 7 unused
% 8 unused
% 9 work value, use this for temporary calculations in a macro

% Boxes
% 0 unused
% 1 used by defer output and macacm
% 2 used by defer output and macacm
% 3 unused
% 4 unused
% 5 unused
% 6 unused
% 7 unused
% 8 unused
% 9 for temp macro use: \boxtop
```



```

% Files for send
% 0   index
% 1   notes
% 2   tofc
% 3   unused
% 4   unused
% 5   unused
% 6   unused
% 7   unused
% 8   unused
% 9   unused

```

The following section consists of external files that are only loaded when needed. As described in the text, this saves on the amount of space needed by these macros in "TEX" itself.

## MBOOK.TEX

```

% Book Format
\def \bookoutput{\vbox to 9truein
  {\baselineskip Opt\lineskipOpt % beginning of output routine, resets skips
  \advpagecount % use the correct page number in \send
  \everyoutput
  \if T\tpage % the next is used when tpage is "T" (title pages)
    {\gdef\tpage{F} % reset tpage
    \vskip .7truein % blank space in place of headlines
    \page} % insert the page contents, no page #
  \else{\if T\index{\indexoutput}
  \else{\if T\pagenumberarea{\pagenumberregion\vfill}\else{}
    \page % insert the page contents
    \if B\pagenumberarea{\vfill\pagenumberregion}\else{}}
  }}} % end \bookOutput routine

\def \pagenumberarea{T} % T for Top of page, B for Bottom, else for none

\def \bookstyle{\maxdepth 2pt
  \parindent 20pt
  \parskip Opt plus 1 pt
  \lineskip 1pt plus Opt
  \topskip 24pt plus 6pt minus 10pt
  \botskip 15pt plus 3pt minus 9pt
  \topbaseline Opt
}

```

```

% page number definitions

\def\boxpagenumberregion{\moveleft .125truein\vbox to .7truein{\hrule
% horizontal rule at top of page
\hbox to 6.75truein{\trule
% 20pt*(1+sqrt(5))/2=32.361pt
\ifeven0{\hbox to 32.361pt{\cmrten\hfill\count0\hfill\trule}
\hfill\cmss12\topmark\hfill}
\else{\hfill\cmss12\botmark\hfill
\hbox to 32.361pt{\cmrten\trule\hfill\count0\hfill}}
\trule}
\hrule}} % horizontal rule under the headline

\def\trule{\vrule height 13.5pt depth 6.5pt} % used at top of page

\def\titemark#1#2{\mark{\ifeven0{#1}\else{#2}}}

\def\pagenumberregion{\boxpagenumberregion}

% "global variables"
\def\tpage{F}
\def\index{F}

\def\titlepage{\gdef\tpage{T}} % \titlepage sets tpage to T

% enable book format

\def\usebookformat{\gdef\standardoutput{\output{\bookoutput}}
\standardoutput
\gdef\parstyle{\bookstyle}
\gdef\normalhsizel{\hsizel 6.5truein}
\gdef\normalvsize{\vsize 8.3truein}
\normal
}

\usebookformat

% chapter section

\def\chapternumber{0}

\def\chapterbegin#1{\par
\gdef\footnotenuml{0}
\advcountq{\chapternumber}
\gdef\sectionnuml{0}
\xdef\wholesectionnuml{Chapter \chapternumber}
\titemark{\wholesectionnuml}{\sectionname}
\vfill\eject
\gdef\sectionname{#1}
\titemark{\wholesectionnuml}{#1}
{\noindent \cmss12 \wholesectionnuml \ \ #1}
\if T\writetofc{\send2{\chapterto{#1}\count0.}}\else{}
\posthdrskip}

```

```

\def\dosectionbegin#1{\par
  \titlemark{\wholesectionnumber}{\sectionname}
  \sectionskip
  \gdef\sectionname{#1}
  \titlemark{\wholesectionnumber}{#1}
  {\tenpoint \bf \noindent $\bullet$\ \wholesectionnumber\ \ #1}
  \posthdrskip}

\def\sectionbegin#1{\advcountq{\sectionnumber}
  \gdef\subsectionnumber{0}
  \xdef\wholesectionnumber{Section \chapternumber.\sectionnumber}
  \if T\writetofc{\send2{\sectiontofc{#1}\count0.}}\else{}
  \dosectionbegin{#1}}

\def\subsectionbegin#1{\advcountq{\subsectionnumber}
  \gdef\dimsectionnumber{0}
  \xdef\wholesectionnumber{Section
    \chapternumber.\sectionnumber.\subsectionnumber}
  \if T\writetofc{\send2{\subsectiontofc{#1}\count0.}}\else{}
  \dosectionbegin{#1}}

\def\dimsectionbegin#1{\advcountq{\dimsectionnumber}
  \xdef\wholesectionnumber{Section
    \chapternumber.\sectionnumber.\subsectionnumber.\dimsectionnumber}
  \if T\writetofc{\send2{\dimsectiontofc{#1}\count0.}}\else{}
  \dosectionbegin{#1}}

\def\specialbegin#1{\titlemark{#1}{\sectionname}
  \vfill\ejct
  \settitle{#1}
  {\noindent \cmss12 #1}
  \posthdrskip}

\def\settitle#1{\par\titlemark{#1}{#1}
  \gdef\wholesectionnumber{#1}
  \gdef\sectionname{#1}}

\def \wholesectionnumber{}
\def \sectionname{}

% automatic table of contents generation

\def\inittofc#1{\open2 #1
  \gdef\writetofc{T} % write tofc info

\def\writetofc{F}

```

## MACACM.TEX

```

% ACM two column format for Versatec

\def\acmoutput{\everypage
\if T\tpage
  {\if T\column
    {\gdef\normalhsz{\hsz 4.25truein}
    \gdef\normalvsize{\vsz 8.9truein}
    \normalhsz\normalvsize
    \save1\page\gdef\column{L}
    }
  \else{\if L\column
    {\save2\page\gdef\column{R}}
    \else{\vbox to 11.9truein{\box1\vskip -1000pt plus 1000000pt
      \hbox to 9 truein{\box2\hfill\page}}
      \advcount 0
      \gdef\column{L}
      \gdef\tpage{F}
      \gdef\normalvsize{\vsz 11.5truein}
      \normalvsize
    }}}
\else{\if L\column
  {\save2\page\gdef\column{R}}
  \else {\vbox to 11.9 truein{\hbox to 9truein{\ninepoint\ifeven0
    {\rm\lastnames\hfill\sl\title}
    \else{\sl\title\hfill\rm\lastnames}}
    \vfill
    \hbox to 9 truein{\box2\hfill\page}}
    \advcount0
    \gdef\column{L}
  }}}

\def \acmstyle{\maxdepth 2pt
  \parindent 20pt
  \parskip 0pt plus 1 pt
  \lineskip 1pt plus 0pt
  \topskip 24pt plus 6pt minus 10pt
  \botskip 15pt plus 3pt minus 9pt
  \topbaseline 0pt
}

\def\endoftitle{\par\vfill\ejct}

\def\endofpaper{\par\vfill\if L\column{\ejct\hbox{}}\vfill}\else{}}

% To use, code \useacmformat at the start of the paper.
% Then define \title and \authors
% then define the title portion, followed by \endoftitle
% when you are all done \endofpaper\end

% enable acm format

\def\standardoutput{\output{\acmoutput}}

```

```

\def\useacmformat{\standardoutput
  \gdef\parstyle{\bookstyle}
  \gdef\normalhsize{\hsize 9truein}
  \gdef\smallhsize{\hsize 9truein} % too small to indent right
  \gdef\normalvsize{\vsize 3truein}
                                % these sizes reuefined in \endoftitle
                                % and \acmoutput

  \normal
  \gdef\tpage{T}
  \gdef\column{T}
}

\useacmformat

```

---

OPLAIN.TEX

```

% Plain Output routine

\def\plainoutput{\advpagecount % use the correct page number in \send
  \page
  \everyoutput}

\def\standardoutput{\output{\plainoutput}}

\def\useplainoutput{\standardoutput
  \gdef\normalhsize{\hsize 6.5truein}
  \gdef\normalvsize{\vsize 9truein}
  \normal
}

\useplainoutput

```

---

OBASIC.TEX

```

% Basic output routine

\def\basicoutput{\advpagecount % use the correct page number in \send
  \vbox to 9truein{\page
    \vfill
    \ctrline{\carten \count0}}
  \everyoutput}

\def\standardoutput{\output{\basicoutput}}

\def\usebasicoutput{\standardoutput
  \gdef\normalhsize{\hsize 6.5truein}
  \gdef\normalvsize{\vsize 8.75truein}
  \normal
}

```

```
\usebasicoutput
```

---

OWOODS.TEX

```
% Woods output
% (To look like previous versions of the annual report.)

\def\Woodsoutput{\advpagecount
  \vbox to 9truein{\ctrline{\ninepoint\sl -\count0-}
    \vfill
    \page}
  \everyoutput}

\def\standardoutput{\output{\Woodsoutput}}

\def\useWoodsoutput{\standardoutput}
  \gdef\normalhsize{\hsize 6.5truein}
  \gdef\normalvsize{\vsize 8.75truein}
  \normal
}

\useWoodsoutput
```

---

SBLOCK.TEX

```
% Block Style

\def\blockstyle{\maxdepth 2pt
  \parindent 0pt
  \parskip 6 pt plus 6 pt minus 2 pt % Skip a line between paragraphs.
  \lineskip 1pt plus 0pt
  \topskip 24pt plus 6pt minus 10pt
  \botskip 15pt plus 3pt minus 9pt
  \topbaseline 0pt
}

\def\useblockstyle{\gdef\parstyle{\blockstyle}
  \normal
}

\useblockstyle
```

## SBASIC.TEX

```
% Basic Style

\def\basicstyle{\maxdepth 2pt
  \parindent 20pt
  \parskip 0pt plus 1 pt
  \lineskip 1pt plus 0pt
  \topskip 24pt plus 6pt minus 10pt
  \botskip 15pt plus 3pt minus 9pt
  \topbaseline 0pt
}

\def\usebasicstyle{\gdef\parstyle{\basicstyle}
  \normal
}

\usebasicstyle
```

## MNOTES.TEX

```
% notes

\open1=fixnot.tex

\def \sendnotes#1{\send1{Page \count0. #1\par}}

\def \putnotes{\specialbegin{Fixup Notations}
  \open1=dummy1.tmp      % Close the fixnot file
  \input fixnot.tex      % Now put text here.
}
```

## DEFER.TEX

```
% defernode based on that written by Brent Hailpern and Jim Boyce

% box 1 is slop on current page
% box 2 is extra slop on current page that will go on following page

\def\defer{\save2\ vbox{}      % no extra slop yet
  \output{\save1\page\output{\save2\page}} % cause stuff to be saved
  \eject                      % flush out current page
  \standardoutput
  \ifdimen 1ht2>0pt{\unbox1\save1\box2}\else{} % comment below
  % put out full page and copy partial page
  \if F\deferredpage{\gdef\deferredpage{T}
    \advppagecountone
    \savethepagecount
  }\else{}
```

```

    \gdef\indefermode{T}
}

\def\enddefer{\eject
  \unbox1
  \gdef\indefermode{F}
}

\def\savethepagecount{\setq{\savedpagecount}9}

\defer % do it this time too!

```

## MTOFC.TEX

```

% table of contents

\def\beginofc{\gdef\chapternumber{0}
  \setpagecount{-1} % initial page number for cover page
  \specialbegin{Table of Contents}}

\def\chapterto#c1#2.{\par
  \advcountq{\chapternumber}
  \gdef\sectionnumber{0}
  \hbox to size{\hbox to 30pt{\bf\chapternumber\hfill}}{#1}
  \leaders\hrule\hfill\hbox to 20pt{\hfill#2}}

\def\sectionto#c1#2.{\par
  \advcountq{\sectionnumber}
  \gdef\subsectionnumber{0}
  \hbox to size{\hbox to 45pt{\bf\chapternumber.\sectionnumber\hfill}}{#1}
  \leaders\hrule\hfill\hbox to 20pt{\hfill#2}}

\def\subsectionto#c1#2.{\par
  \advcountq{\subsectionnumber}
  \gdef\dimsectionnumber{0}
  \hbox to size{\hbox to 60pt{\bf
    \chapternumber.\sectionnumber.\subsectionnumber\hfill}}{#1}
  \leaders\hrule\hfill\hbox to 20pt{\hfill#2}}

\def\dimsectionto#c1#2.{\par
  \advcountq{\dimsectionnumber}
  \hbox to size{\hbox to 75pt{\bf
    \chapternumber.\sectionnumber.\subsectionnumber.\dimsectionnumber
    \hfill}}{#1}
  \leaders\hrule\hfill\hbox to 20pt{\hfill#2}}

\def\endtofc{\par\vfill\eject % put out this page before screwing up page #
  \gdef\chapternumber{0}
  \setpagecount{0}
}

\beginofc % do it now too!

```



## NOFILL Program

## NOFILL Program

Lynne A. Price  
Patrick Milligan

BNR INC.,  
subsidiary of Bell-Northern Research, Ltd.

Below is the source for a SAIL program called NOFILL. This program reads an ASCII text file and outputs a file that can be processed by TeX in order to typeset a listing of the original file. Line spacing is preserved and characters which have special meaning to TeX (*e.g.*, backslash and braces) do not cause problems. This program is very useful to documenters of TeX macros who wish to prepare lengthy examples of corresponding input and output. Preparation of documents that include sample programs in various programming languages is also simplified with this program. For example, the listing shown here was itself generated by NOFILL. Following the program listing is some test data that can be used to verify this code and the corresponding output.

Source of NOFILL:

```
begin "nofill"
comment This program generates a TeX input file that typesets ASCII
text files (e.g., TeX macro source). The output file has the form
    {\ty\saves9\hbox{A}
    \def \# {{\ty\char`043}} % Defines for special characters
    .
    .
    \def \up {{\ty\char`136}}
    \hbox{first line}
    \hbox{second line}
    .
    .
    \hbox{last line}
    \hbox{}
}
```

The call to \ty on the first line is assumed to select an appropriate font, usually a fixed-width typewriter font. The width of any character (the letter "A", for instance) can then be used to space special characters from other fonts. Box 9 is used to save this width.

The text of each line is copied directly from the input file to the output

file with special characters changed to macro calls (for example, a space in the input is output as "\ " and a backslash is output as "\\"). When long input lines occur, or when an input line contains several special characters that expand to macros with long names, one \hbox call may be output over several lines in the output file. The macros used to output special characters are controlled by the SAIL macro "chartable" defined below. Users at different sites or with different applications may wish to modify this table.

```

;

define crlf = "`15&`12";
define cr = "`15" ; define lf = "`12" ;
define normal = "0", foundcr = "1", foundtab = "2", special = "3" ;

comment Output lines are broken after maxlen characters (or after the
first control sequence that extends past maxlen characters). This cutoff
makes output files easier to read on terminals with narrow screens and
also prevents the generation of TeX input lines longer than the maximum
150 characters;
define maxlen = "60" ;

require "<><" delimiters ;
comment The following table defines the processing performed on each input
character. The SAIL macro "action" has three parameters: the name of the
ASCII character, the name of an action to be performed when the character
is encountered (used to control the case statement in the main program loop),
and, for special characters, a string to be output when the character is
encountered. The order of actions in this table is significant--when
changes are made, the order must be preserved.
;
define chartable=<
action(<NUL>,<special>,<"\up @">),
action(<^A (SOH)>,<special>,<"\up A">),
action(<^B (STX)>,<special>,<"\up B">),
action(<^C (ETX)>,<special>,<"\up C">),
action(<^D (EOT)>,<special>,<"\up D">),
action(<^E (ENQ)>,<special>,<"\up E">),
action(<^F (ACK)>,<special>,<"\up F">),
action(<^G (BEL)>,<special>,<"\up G">),
action(<^H (BS)>,<special>,<"\up H">),
action(<^I (HT)>,<foundtab>,<"\up I">),
action(<^J (LF)>,<special>,<"\up J">),
action(<^K (VT)>,<special>,<"\up K">),
action(<^L (FF)>,<special>,<"\up L">),
action(<^M (CR)>,<foundcr>,<"\up M">),
action(<^N (SO)>,<special>,<"\up N">),
action(<^O (SI)>,<special>,<"\up O">),
action(<^P (DLE)>,<special>,<"\up P">),
action(<^Q (DC1)>,<special>,<"\up Q">),
action(<^R (DC2)>,<special>,<"\up R">),
action(<^S (DC3)>,<special>,<"\up S">),
action(<^T (DC4)>,<special>,<"\up T">),
action(<^U (NAK)>,<special>,<"\up U">),

```

## NOFILL Program

```

action(<~V (SYN)>,<special>,<"\up V">),
action(<~W (ETB)>,<special>,<"\up W">),
action(<~X (CAN)>,<special>,<"\up X">),
action(<~Y (EM)>,<special>,<"\up Y">),
action(<~Z (SUB)>,<special>,<"\up Z">),
action(<ESC>,<special>,<"\up [">),
action(<FS>,<special>,<"\up \\">),
action(<GS>,<special>,<"\up ]">),
action(<RS>,<special>,<"\up \^">),
action(<US>,<special>,<"\up \_">),
action(<space>,<special>,<"\ ">),
action(<!>,<normal>,<">),
action(<">,<normal>,<">),
action(<#>,<special>,<"#>),
action(<$>,<special>,<"$>),
action(<%>,<special>,<"%>),
action(<_>,<normal>,<">),
action(<`>,<special>,<"\">),
action(<(>,<normal>,<">),
action(<>>,<normal>,<">),
action(<*>,<normal>,<">),
action(<+>,<normal>,<">),
action(<,>,<normal>,<">),
action(<->,<normal>,<">),
action(<.>,<normal>,<">),
action(</>,<normal>,<">),
[10] action(<digits>,<normal>,<">),
action(<:>,<normal>,<">),
action(<:>,<normal>,<">),
action(<less than>,<normal>,<">),
action(<=>,<normal>,<">),
action(<greater than>,<normal>,<">),
action(<?>,<normal>,<">),
action(<@>,<normal>,<">),
[26] action(<upper case letters>,<normal>,<">),
action(<[>,<normal>,<">),
action(<\>,<special>,<"\">),
action(<]>,<normal>,<">),
action(<~>,<special>,<"\">),
action(<_>,<special>,<"\">),
action(<~>,<special>,<"\">),
[26] action(<lower case letters>,<normal>,<">),
action(<{>,<special>,<"\">),
action(<|>,<special>,<"\">),
action(<}>,<special>,<"\">),
action(<~>,<special>,<"\">),
action(<DEL>,<special>,<"\up ?">
>;
define action(a,b,c) = <b>;
preload!Xwith chartable;
integer array states[0:127] ;
redefine action(a,b,c) = <c> ;
preload!Xwith chartable;

```

```

string array strings[0:127] ;

external integer !SKIP! ;

string infile, outfile, nextchar;
integer chan, dbreak, deof, table, state, outcnt, incnt, i, spacestotab ;
boolean skipnext ;

procedure cntprint(string s); comment Output a string and count its length;
begin
  outcnt := outcnt + length(s) ;
  print(s) ;
end ;

!SKIP! := TRUE ;
print("NOFILL",crlf) ;
while !SKIP! do begin
  print ("Input file name? ");
  infile := intty;
  lookup(chan,infile,deof);
  chan := openfile(infile, "ROE") ;
  if !SKIP! then begin
    infile := infile&".TEX";
    lookup(chan,infile,deof);
    chan := openfile(infile, "ROE") ;
    if !SKIP! then print(infile," bad. Try again...",crlf);
  end ;
end;
setinput(chan,1,dbreak, deof);
print("Output file (Default: NOFILL.TEX) ? ");
if (outfile:=intty) = "" then outfile:="NOFILL.TEX" ;
setprint(outfile,"F") ;

print("{\ty",crlf) ;
print("\def \#  {\ty\char`043}}", crlf) ;
print("\def \$  {\ty\char`044}}", crlf) ;
print("\def \%  {\ty\char`045}}", crlf) ;
print("\def \`  {\ty\char`015}}", crlf) ;
print("\def \\  {\ty\char`134}}", crlf) ;
print("\def \^  {\ty\char`017}}", crlf) ;
print("\def \_  {\ty\char`032}}", crlf) ;
print("\def \^  {\ty\char`016}}", crlf) ;
print("\def \{  {\ty\char`173}}", crlf) ;
print("\def \|  {\ty\char`174}}", crlf) ;
print("\def \}  {\ty\char`176}}", crlf) ;
print("\def \~  {\ty\char`024}}", crlf) ;

print("\def \up {\ty\char`136}}", crlf) ;

print(crlf, "\hbox{") ;

setbreak(table := getbreak,"", "", "I") ;
deof := 0 ;

```

## NOFILL Program

```

nextchar := input(chan, table) ;
incnt:= 1 ; comment incnt counts characters on input line ;
outcnt:= length("\hbox{") ; comment outcnt counts characters on output line ;
while NOT deof do begin
  if outcnt > maxlen then begin
    print("\!" & crlf) ;
    outcnt := 0 ;
  end ;
  skipnext := FALSE ; comment used for "lookahead" in cr-lf pairs ;
  case states[nextchar] of begin
    [normal]
    begin
      cntprint(nextchar) ;
    end ;
    [special]
    begin
      cntprint(strings[nextchar]) ;
    end ;
    [foundcr]
    begin
      nextchar := input(chan, table) ;
      if nextchar = lf then begin
        print(")", crlf, "\hbox{") ;
        incnt := 0 ;
        outcnt := length("\hbox{") ;
      end
      else begin
        cntprint(strings[cr]) ;
        skipnext := TRUE ;
      end ;
    end ;
    [foundtab]
    begin
      spacestotab := 8 - ((incnt - 1) MOD 8) ;
      for i := 1 step 1 until spacestotab do cntprint("\ ") ;
      incnt := incnt + spacestotab - 1 ;
    end
  end ;
  if NOT skipnext then nextchar := input(chan, table) ;
  incnt := incnt + 1 ;
end ;
print("}");
end "nofill";

```

Test data for NOFILL:

ASCII Test:

```

^A^B^C^D^E^F^G^H      ^J^K^L^M^N^O^P^Q^R^S^T^U^V^W^X^Y^Z^[^\]^_
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNQRSTUUVWXYZ[\]^_
`abcdeifghijklmnopqrstuvwxyz{|}~?

```

Tab test:

```

12345678901234567890123456789012345678901234567890
.      .      .      .      .      .      .      (tabs)
a      b      c      d      e      f      g
aa     bb     cc     dd     ee     ff     gg
aaa    bbbb   cccc   dddd   eeee   ffff   gggg
aaaa   bbbbb  ccccc  ddddd  eeeee  fffff  ggggg
aaaaa  bbbbbb cccccc dddddd eeeeee ffffff gggggg
aaaaaa bbbbbb cccccc dddddd eeeeee ffffff gggggg
.      .      .      .      .      .      .      (spaces and tabs)
a      b      c      d      e      f      g
 a     b     c     d     e     f     g
  a    b    c    d    e    f    g
   a   b   c   d   e   f   g
    a  b  c  d  e  f  g
     a b c d e f g
.      .      .      .      .      .      .      (spaces)
.      .      .      .      .      .      .      (multiple tabs)

```

Test data output from NOFILL:

```

{\ty
\def \# {{\ty\char`043}}
\def \$ {{\ty\char`044}}
\def \% {{\ty\char`045}}
\def \` {{\ty\char`015}}
\def \\ {{\ty\char`134}}
\def \^ {{\ty\char`017}}
\def \_ {{\ty\char`032}}
\def \' {{\ty\char`016}}
\def \{ {{\ty\char`173}}
\def \| {{\ty\char`174}}
\def \} {{\ty\char`176}}
\def \~ {{\ty\char`024}}
\def \up {{\ty\char`136}}

\hbox{ASCII\ Test:}
\hbox{}
\hbox{\up A\up B\up C\up D\up E\up F\up G\up H\ | | | | | | | | | |
\up J\up K\up L\up M\up N\up O\up P\up Q\up R\up S\up T\up U\up V\up
\up W\up X\up Y\up Z\up [\up \\\up ]\up \~\up \_}
\hbox{\ !"\#\$\%&\'()*+,-./0123456789:;<=>?}
\hbox{@ABCDEFGHIJKLMNopqrstuvwxyz[\]\^_}
\hbox{\`abcdefghijklmnopqrstuvwxyz\{|\}\~\up ?}
\hbox{}
\hbox{}

```



Pascal NOFILL

Page 1

```
(* Generate TEX input file to typeset ASCII text (i.e. TEX macro source) *)
(* Written using Hedrick's TOPS-20 Native Mode Pascal compiler *)
```

```
Program NoFill(Input,Output);
```

```
Const
```

```
  MaxLen = 80;           (* Maximum length of output line *)
  LF = 012B;           (* ASCII Line Feed *)
```

```
Type
```

```
  CharClass = (Printing, Quoted, Control, HTab, Return);
```

```
Var
```

```
  NextChar:   Char;
  CharType:   Array [0..177B] of CharClass;
  InCount:    Integer;
  OutCount:   Integer;
  I:          Integer;      (* Scratch counter *)
```

```
Procedure Initialize;      (* Initialize CharType array... *)
```

```
Begin (*Initialize*)
```

```
  (* Start of ASCII Control Characters: *)
```

```
  For I := 000B to 010B Do      (* NUL to BS *)
    CharType[I] := Control;
```

```
  CharType[011B] := HTab;      (* TAB *)
```

```
  CharType[012B] := Control;   (* LF *)
```

```
  CharType[013B] := Control;   (* VT *)
```

```
  CharType[014B] := Control;   (* FF *)
```

```
  CharType[015B] := Return;    (* CR *)
```

```
  For I := 016B to 037B Do      (* SO to US *)
    CharType[I] := Control;
```

```
  (* End of ASCII Control Characters *)
```

```
  CharType[040B] := Quoted;    (* Space *)
```

```
  CharType[041B] := Printing;  (* ! *)
```

```
  CharType[042B] := Printing;  (* " *)
```

```
  CharType[043B] := Quoted;    (* # *)
```

```
  CharType[044B] := Quoted;    (* $ *)
```

```
  CharType[045B] := Quoted;    (* % *)
```

```
  CharType[046B] := Printing;  (* & *)
```

```
  CharType[047B] := Quoted;    (* ` *)
```

```
  For I := 050B to 132B Do      (* ( to Z *)
    CharType[I] := Printing;
```



```

CharType[133B] := Printing;      (* [ *)
CharType[134B] := Quoted;       (* \ *)
CharType[135B] := Printing;     (* ] *)
CharType[136B] := Quoted;       (* ^ *)
CharType[137B] := Quoted;       (* _ *)
CharType[140B] := Quoted;       (* ` *)

```

```

For I := 141B to 172B Do        (* a to z *)
  CharType[I] := Printing;

```

```

CharType[173B] := Quoted;      (* { *)
CharType[174B] := Quoted;      (* | *)
CharType[175B] := Quoted;      (* } *)
CharType[176B] := Quoted;      (* ~ *)
CharType[177B] := Control;     (* DEL *)

```

```
End; (*Initialize*)
```

```
Procedure OutChar(InChar: Char);
```

```
Var
```

```

Spaces:   Integer;      (* Spaces needed to expand a TAB *)
NextChar: Char;         (* Used for lookahead in CR-LF pairs *)

```

```
Begin (*Output*)
```

```
InCount := InCount + 1;
```

```
Case CharType[Ord(InChar)] of
```

```

  Printing: Begin
    Write(InChar);
    OutCount := OutCount + 1;
  End;

```

```

  Quoted: Begin
    Write(`\`, InChar);
    OutCount := OutCount + 2;
  End;

```

```

  Control: Begin
    Write(`\up `);
    OutCount := OutCount + 4;
    InCount := InCount - 1;
    If Ord(InChar) > 100B Then Begin (* DEL is Special *)
      OutChar(Chr(Ord(InChar) - 100B));
    End Else Begin
      OutChar(Chr(Ord(InChar) + 100B));
    End;
  End;
End;

```

```

      HTab:      Begin
                  Spaces := 8 - ((InCount - 1) MOD 8);
                  For I := 1 to Spaces Do Write('\ ');
                  OutCount := OutCount + 2*Spaces;
                  InCount := InCount + Spaces - 1;
      End;

      Return:    Begin
                  Read(NextChar);
                  If Ord(NextChar) = LF Then Begin
                      WriteLn('^');
                      Write('\hbox{');
                      OutCount := 6;      (* Length of \hbox{ *)
                      InCount := 0;
                  End Else Begin
                      Write('\up M');
                      OutCount := OutCount + 5;
                      OutChar(NextChar);
                  End;
      End;

      End; (*Case*)

      End; (*OutChar*)

      Begin (*NoFill*)

          Initialize;      (* Set up CharType array *)

          WriteLn('^{\ty}');

          WriteLn('\def \# {{\ty\char``043}}');
          WriteLn('\def \$ {{\ty\char``044}}');
          WriteLn('\def \% {{\ty\char``045}}');
          WriteLn('\def \^ {{\ty\char``015}}');
          WriteLn('\def \_ {{\ty\char``134}}');
          WriteLn('\def \~ {{\ty\char``017}}');
          WriteLn('\def \_ {{\ty\char``032}}');
          WriteLn('\def \^ {{\ty\char``016}}');
          WriteLn('\def \{ {{\ty\char``173}}');
          WriteLn('\def \| {{\ty\char``174}}');
          WriteLn('\def \} {{\ty\char``176}}');
          WriteLn('\def \^ {{\ty\char``024}}');

          WriteLn('\def \up {{\ty\char``136}}');

          Write('\hbox{');
          OutCount := 6;      (* Length of \hbox{ *)
          InCount := 0;

          While Not(Eof(Input)) Do
              Begin

```

```
    If OutCount > MaxLen Then Begin
      WriteLn(`\!`);
      OutCount := 0;
    End;
```

```
    Read(NextChar);
    OutChar(NextChar);
```

```
  End;
```

```
WriteLn(`}}`);
```

```
End.
```

## LIST MACROS

Lynne A. Price

BNR INC.,  
 subsidiary of Bell-Northern Research, Ltd.

In many document systems, there is a need for lists of indented paragraphs marked by numbers, bullets, letters, or other symbols. This article describes the design of a TeX macro package that provides such a facility with examples of the use of the macros and a listing of their source.

- An author indicates in a TeX input file that a list is to begin with a call to a start-of-list macro. The start-of-list macros indicate the type of list.
  - (1) Lists begun with the following macros are marked with numbers or letters enclosed in parentheses:
    - (a) `\numberlist` — Arabic numerals
    - (b) `\romanlist` — lower-case Roman numerals
    - (c) `\ROMANLIST` — upper-case Roman numerals
    - (d) `\alphalist` — lower-case letters
    - (e) `\ALPHALIST` — upper-case letters
  - (2) Lists begun with the following macros are marked with numbers or letters followed by a period:
    - a. `\dotnumberlist` — Arabic numerals
    - b. `\dotromanlist` — lower-case Roman numerals
    - c. `\DOTROMANLIST` — upper-case Roman numerals
    - d. `\dotalphalist` — lower-case letters
    - e. `\DOTALPHALIST` — upper-case letters
  - (3) Lists begun with the following macros are marked with numbers or letters that have no surrounding punctuation:
    - a `\nopuncnumberlist` — Arabic numerals
    - b `\nopuncromanlist` — lower-case Roman numerals
    - c `\NOPUNCROMANLIST` — upper-case Roman numerals
    - d `\nopuncalphalist` — lower-case letters
    - e `\NOPUNCALPHALIST` — upper-case letters
  - (4) Lists begun with the following macros are marked with numbers or letters followed by a

right parenthesis:

- a) `\closenumberlist` — Arabic numerals
- b) `\closeromanlist` — lower-case Roman numerals
- c) `\CLOSEROMANLIST` — upper-case Roman numerals
- d) `\closealphalist` — lower-case letters
- e) `\CLOSEALPHALIST` — upper-case letters

- (5) The macro `\bulletlist` starts a list of items marked by bullets.
  - (6) The macro `\dashlist` starts a list of items marked by em-dashes.
  - (7) The macro `\marklist <mark>` starts a list in which each item is marked by the string specified as `<mark>`.
- A call to the macro `\listitem` should precede each item (including the first one) on a list except when the list items occur within tables. This macro inserts the list mark specified by the start-of-list macro, causes the following text to appear in indented blocked paragraphs, and increments numeric and alphabetic marks.
  - The user may specify a mark for a particular item by using the macro `\markitem <mark>` instead of `\listitem`. Lists in which every mark is specified in this fashion may be begun with `\startlist` instead of one of the start-of-list macros enumerated above. **Warning:** Spaces and carriage returns following macro names are insignificant; in fact, when a macro such as `\listitem` has no parameters, unless the following text begins with a special character, a space or carriage return is required to delimit the macro name. However, the TeX spacing conventions cause a space or carriage return after the right brace that ends the mark specified with `\markitem` to result in additional space between the mark and the beginning of the list item. As a result of these conventions the input

```
\listitem
Note the position of the first word here.
```

is equivalent to

```
\listitem Note the position of the first word here.
```

while

```
\listitemNote the position of the first word here.
```

would result in an error, because the macro `\listitemNote` has not been defined. However, while

```
\markitem{A---}
Note the position of the first word here.
```

is equivalent to

```
\markitem{A---} Note the position of the first word here.
```

the two latter probably do not produce what the user intended and are not equivalent to the correct form

```
\markitem{A---}Note the position of the first word here.
```

- Lists may occur within other lists—up to four levels of nesting are permitted.
- The macro `\listmark` can be used within tables to insert the current list mark. Like `\listitem`, `listmark` increments numeric and alphabetic marks. It does not, however, affect the current

**List Macros**

paragraph structure.

- The end of a list is indicated by the macro `\endlist`.
- The TeX control sequence `\par` used to indicate the end of paragraphs should not be included before the beginning of a list or before calling `\listitem`. However, individual items may contain several paragraphs separated by calls to `\par`.
- **Warning:** A blank line or a line containing only a comment is treated by TeX identically to the control sequence `\par`. Normally, several adjacent calls to `\par` are equivalent to a single call, so that one or more blank lines or comment lines can appear between paragraphs. Within lists, however, `\par` is redefined so that successive paragraphs will have the appropriate indentation. Multiple calls to `\par` within lists can create unexpected results. Therefore, blank lines and comment lines should be avoided within lists.

These features are illustrated in the following example taken from Acts II and III of *Hamlet*:

```

What a piece of work is a man! How noble in reason!
How infinite in faculty, in form and moving! How express and
admirable in action! How like an angel in apprehension!
How like a god! The beauty of the world! The paragon of animals!
And yet, to me, what is this quintessence of
dust? $\ldots$ \numberlist \listitem To be,
or not to be: that is the question. Whether:
\alphalist \listitem
`Tis nobler in the mind to suffer the slings and arrows of
outrageous fortune; or
\listitem To take arms against a sea of troubles,
and by opposing, end them. \endlist
\listitem
To die; to sleep; no more;
\alphalist \listitem
And by a sleep to say we end the heart-ache and the thousand natural shocks
that flesh is heir to.
\listitem
`Tis a consumation devoutly to be wish'd. \endlist \listitem
To die; to sleep;---to sleep?
Perchance to dream! Ay, there's the rub.
For in that sleep of death what dreams may come, when we have shuffl'd off
this mortal coil, must give us pause. There's the respect
that makes calamity of so long life. \endlist
$\ldots$ Thus conscience does make cowards of us all; and thus the native hue
of resolution is sicklied o'er with the pale cast of thought, and enterprises
of great faith and moment with this regard their currents turn awry, and lose
the name of action.

```

Figure 1. Nested Lists—Sample Input

## List Macros

What a piece of work is a man! How noble in reason! How infinite in faculty, in form and moving! How express and admirable in action! How like an angel in apprehension! How like a god! The beauty of the world! The paragon of animals! And yet, to me, what is this quintessence of dust?...

- (1) To be, or not to be: that is the question. Whether:
  - (a) 'Tis nobler in the mind to suffer the slings and arrows of outrageous fortune; or
  - (b) To take arms against a sea of troubles, and by opposing, end them.
- (2) To die; to sleep; no more;
  - (a) And by a sleep to say we end the heart-ache and the thousand natural shocks that flesh is heir to.
  - (b) 'Tis a consumation devoutly to be wish'd.
- (3) To die; to sleep;—to sleep? Perchance to dream! Ay, there's the rub. For in that sleep of death what dreams may come, when we have shuff'd off this mortal coil, must give us pause. There's the respect that makes calamity of so long life.

... Thus conscience does make cowards of us all; and thus the native hue of resolution is sicklied o'er with the pale cast of thought, and enterprises of great faith and moment with this regard their currents turn awry, and lose the name of action.

Figure 2. Nested Lists—Sample Output

The example below shows the use of the list macros in conjunction with TeX's alignment feature (the latter is used to produce tables). Alignment is described in Chapter 22 of the TeX manual.



```

\numberlist
\ctrline{Menu}
\ctrline{June 25, 1980}
\vskip 4ex
\ctrline{\vbox{\halign{\hfill#\quad|#\hfill}\cr
\listmark|Tomato Bisque Soup\cr
\listmark|Spinach Salad\cr
\listmark|Tuscan Pot Roast\cr
\listmark|Zucchini Souffl\`e\cr
\listmark|Rice Pilaf\cr
\listmark|Braided Onion Bread\cr
\listmark|Cream Puff Swans\cr
}}
\endlist

```

Figure 3. Lists Within Tables—Sample Input

Menu  
June 25, 1980

- (1) Tomato Bisque Soup
- (2) Spinach Salad
- (3) Tuscan Pot Roast
- (4) Zucchini Soufflé
- (5) Rice Pilaf
- (6) Braided Onion Bread
- (7) Cream Puff Swans

Figure 4. Lists Within Tables—Sample Output

The remainder of this article describes details about the list macros that many readers may wish to skip.

- The indentation for all levels of nested lists is determined by the font in effect when the *outermost* list is started. Each level of list is indented by an amount of space equivalent to 4 ems in the initial font. This default may be changed by setting `\varunit <dimen>` where `<dimen>` is the amount of indentation desired. (See Figures 5 and 6 below.)
- The list macros use the single “variable unit” provided by TeX. Therefore, the user must not redefine `\varunit` except to change the indentation of list items as described above.
- Since TeX’s `\par` control sequence is used to end a paragraph rather than to start a new one, `\par`

## List Macros

should not precede the text of list items. Nevertheless, to ensure proper indentation within lists, a macro call is necessary at the beginning of each paragraph. The required spacing information is provided at the beginning of list items by the macros `\listitem` and `\markitem {<mark>}` and by `\par` before succeeding paragraphs. A special case occurs, however, after the `\endlist` that follows a list that appears within another list when the following text is part of the same list item in the outer list as the one that contained the inner list. In this situation, the `\endlist` that ends the inner list should be immediately followed by `\continue`. However, the `\endlist` that ends the outermost list can be immediately followed by the text of the following paragraph. These situations (and the `\varunit` construct discussed above) are illustrated in the following example:

```
\def\trademark{\raise1.4ex\hbox{\spose{\raise 1ex\hbox{\curfont @\char`142}}}\!
\hbox{\curfont r}}}
```

TSR Games manufactures rule books, character records, polyhedral dice, and other accessories for playing the fantasy role-playing game, `{\it Dungeons and Dragons}`\trademark.

Each player creates a character whose traits are determined by rolling dice. The resulting characters belong to one of several classes including:

```
\startlist
\save9\hbox{\it Magic Users:\quad\quad} % Set list indentation to the width of
\varunit 1wd9 % the string "Magic Users: "
\markitem{\it Fighters:}Any human character can be a fighter as are most
halflings, dwarves and elves. Fighters
\marklist(*)\listitem
Can use any weapon.\listitem Can wear armor, including magic armor.
\listitem Can do no magic.\listitem Become harder
to kill as they become more experienced.\endlist
\markitem{\it Thieves:}Human characters can be thieves. Thieves have
special abilities:\bulletlist\listitem
They can survive attacks from behind.
\listitem They can climb sheer surfaces.
\listitem They can pick locks and pockets.
\endlist % the following is part of list item containing entire preceding list
\continue
```

Special rules exist for halflings, dwarves, and elves who are thieves.

```
\markitem{\it Magic Users:}Any human can also be a magic user.
Magic users cannot wear armor or use most magical weapons.
They can, however, use all other magic items and they can cast spells.
\endlist
```

The game is played as a series of adventures. Players may play the same character for several adventures, and the character gains in experience as it survives each adventure.

Figure 5. Use of `\continue`—Sample Input

TSR Games manufactures rule books, character records, polyhedral dice, and other accessories for playing the fantasy rôle-playing game, *Dungeons and Dragons*®. Each player creates a character whose traits are determined by rolling dice. The resulting characters belong to one of several classes including:

**Fighters:** Any human character can be a fighter as are most halflings, dwarves and elves. Fighters

- \* Can use any weapon.
- \* Can wear armor, including magic armor.
- \* Can do no magic.
- \* Become harder to kill as they become more experienced.

**Thieves:** Human characters can be thieves. Thieves have special abilities:

- They can survive attacks from behind.
- They can climb sheer surfaces.
- They can pick locks and pockets.

Special rules exist for halflings, dwarves, and elves who are thieves.

**Magic Users:** Any human can also be a magic user. Magic users cannot wear armor or use most magical weapons. They can, however, use all other magic items and they can cast spells.

The game is played as a series of adventures. Players may play the same character for several adventures, and the character gains in experience as it survives each adventure.

Figure 6. Use of \continue—Sample Output

- Block paragraph structure is used inside lists. At the end of a list, paragraph indentation is reset according to the value specified in the macro \saveparindent. Users of standard macro packages need not concern themselves with the latter. Users, however, who design their own formats should know that the default value is 2 ems in the current font. To select another value, use

```
\def\saveparindent{<dimen>}
```

The slide macros, for example, which assume block paragraph structure throughout, include the command

```
\def\saveparindent{0 pt}
```

(here 0 is the number zero not the letter O).

## List Macros

- Since each list item is assumed to start a new paragraph, the same amount of vertical space that TeX inserts between paragraphs is inserted between list items. When the list items are very short, this amount of white space is excessive. The space between paragraphs is controlled with the TeX command `\parskip <dimen>`, where, following Knuth's notation, `<dimen>` may be any dimension or size. For example, `\parskip Opt` completely suppresses extra spacing between list items. Figure 7 illustrates how this command was used at the beginning of this article. Note the use of TeX's grouping feature (braces) to indicate that the change in `\parskip` should have effect only within the inner lists.

```

\numberlist\listitem
Lists begun with the following macros are marked with numbers or letters
enclosed in parentheses:
\alphalist
\listitem{\ty \numberlist} --- Arabic numerals
{\parskip Opt
\listitem{\ty \romanlist} --- lower-case Roman numerals
\listitem{\ty \ROMANLIST} --- upper-case Roman numerals
\listitem{\ty \alphalist} --- lower-case letters
\listitem{\ty \ALPHALIST} --- upper-case letters
\endlist}
\listitem
Lists begun with the following macros are marked with numbers or letters
followed by a period:
\dotalphalist
\listitem{\ty \dotnumberlist} --- Arabic numerals
{\parskip Opt
\listitem{\ty \dotromanlist} --- lower-case Roman numerals
\listitem{\ty \DOTROMANLIST} --- upper-case Roman numerals
\listitem{\ty \dotalphalist} --- lower-case letters
\listitem{\ty \DOTALPHALIST} --- upper-case letters
\endlist}
\endlist

```

Figure 7. Changing the Spacing Between List Items—Samp'

- (1) Lists begun with the following macros are marked with numbers or letters enclosed in parentheses:
- (a) `\numberlist` — Arabic numerals
  - (b) `\romanlist` — lower-case Roman numerals
  - (c) `\ROMANLIST` — upper-case Roman numerals
  - (d) `\alphalist` — lower-case letters
  - (e) `\ALPHALIST` — upper-case letters
- (2) Lists begun with the following macros are marked with numbers or letters followed by a period:
- a. `\dotnumberlist` — Arabic numerals
  - b. `\dotromanlist` — lower-case Roman numerals
  - c. `\DOTROMANLIST` — upper-case Roman numerals

Figure 8. Changing the Spacing Between List Items—Sample Output

The TeX source of a set of list macros to implement this design is shown below:

```
% List Macros

% \neg and \ifzero from Appendix X
\def\neg#1{\setcount#1-\count#1}
\def\ifzero#1#2\else#3{\ifpos#1{#3}\else{\neg#1
  \ifpos#1{\neg#1 #3}\else{\neg#1 #2}}
% Arguments to \ifeq can be constants or counters
\def\ifeq#1#2#3\else#4{\setcount9 #1 \advcount9 by -#2
  \ifzero9{#3}\else{#4}}

% Count 9 for scratch, count 8 for list level, count 7 for current item number
% Box 9 for scratch, File 9 for error messages

% \Alph prints the value of the specified counter according to
% the alphabetic sequence A, B, C, ... I.e., if the value of the
% counter is 1, \Alph prints A, etc. \alph does the same except
% it generates lower case instead of upper case letters.
\def\Alph#1{\setcount9 \count#1 \advcount9 by `100 \char\count9}
\def\alph#1{\setcount9 \count#1 \advcount9 by `140 \char\count9}

% Make \endpar a synonym for TeX's standard \par control sequence.
% Within a list, \par is redefined to produce an indented paragraph.
\let \endpar=\par

% TeX counter 8 is used to count list indentation level. \ifcounteight
% executes its ith argument if the current value of counter 8 is i.
% Note that only four levels of list are permitted.
\def \ifcounteight#1#2#3#4{
  \ifeq1{\count8}{#1}
```

## List Macros

```

\else{\ifeq2{\count8}{#2}
  \else{\ifeq3{\count8}{#3}
    \else{\ifeq4{\count8}{#4}\else{}}}}

% \startpar starts a paragraph inside a list
\def \startpar{\hangindent \count8vu$ $}

% Start a new paragraph in a current list item after the end of a nested
% list
\def \continue{\startpar\hbox to \count8vu{}}

% After a list, paragraph indentation is reset according to \saveparindent
% By default, this value is 2em
\def\saveparindent{2em}

\setcount8 0
\parskip 2ex

% At start of an indented list, save item number of outer list
\def \savecount{
  \ifcounteight{\xdef\savea{\count7}}{\xdef\saveb{\count7}}{\xdef
  \savec{\count7}}{\xdef\saved{\count7}}

% At end of an indented list, restore item number from previous level
\def \restorecount{
  \ifcounteight{\setcount7 \savea}{\setcount7 \saveb}{\setcount7
  \savec}{\setcount7 \saved}}

% Initially, no current list active
\def\listitemerror{\send9{LIST ITEM ENCOUNTERED BUT NO CURRENT LIST}}
\def\listmarkerror{\send9{LIST MARK ENCOUNTERED BUT NO CURRENT LIST}}
\def \listitem{\listitemerror}
\def \listmark{\listmarkerror}

% Provide for converting item numbers to upper or lower case roman
\def\roman#1{\setcount9 -\count#1\count9}
\def\Roman#1{\setcount 9
  -\count#1\xdef\uppercaseroman{\uppercase{\count9}}\uppercaseroman}

\def \markitem#1{\setmark{#1}\listitem}
\def \setmark#1{
  \ifcounteight{\gdef \marka{#1}}{\gdef \markb{#1}}{\gdef
  \markc{#1}}{\gdef \markd{#1}}

% Start of list macros
\def \numberlist{
  \startlist
  \setmark{(\count7)}}
\def \alphalist{
  \startlist
  \setmark{(\alph7)}}
\def \Alphalist{
  \startlist

```

```

\setmark{\Alph7}}
\def \romanlist{
\startlist
\setmark{\roman7}}
\def \Romanlist{
\startlist
\setmark{\Roman7}}
\def \dotnumberlist{
\startlist
\setmark{\count7.}}
\def \dotalphalist{
\startlist
\setmark{\alph7.}}
\def \DOTAlphalist{
\startlist
\setmark{\Alph7.}}
\def \dotromanlist{
\startlist
\setmark{\roman7.}}
\def \DOTRomanlist{
\startlist
\setmark{\Roman7.}}
\def \nopuncnumberlist{
\startlist
\setmark{\count7}}
\def \nopuncalphalist{
\startlist
\setmark{\alph7}}
\def \NOPUNCAphalist{
\startlist
\setmark{\Alph7}}
\def \nopuncromanlist{
\startlist
\setmark{\roman7}}
\def \NOPUNCRomanlist{
\startlist
\setmark{\Roman7}}
\def \closenumberlist{
\startlist
\setmark{\count7}}
\def \closealphalist{
\startlist
\setmark{\alph7}}
\def \CloseAlphalist{
\startlist
\setmark{\Alph7}}
\def \closeromanlist{
\startlist
\setmark{\roman7}}
\def \CLOSERomanlist{
\startlist
\setmark{\Roman7}}
\def \bulletlist{

```

## List Macros

```

\startlist
\setmark{${bullet$}}
\def \dashlist{
\startlist
\setmark{\rm ---}}
\def \marklist#1{
\startlist
\setmark{#1}}

% Start a list
\def \startlist{
\advcount8\setcount 9 \count8 \advcount9 by -4 % Compute level number
\ifpos9{
\ldef\listerror{ATTEMPT TO START \count8TH LEVEL OF NESTED LISTS ---
ONLY 4 LEVELS ALLOWED}
\send9{\listerror}}
\else{ % Compute indent if first level
\ifeq1{\count8}{\save9\hbox{0}\varunit 4wd9}\else{}
\savecount
\setcount7 0
\parindent Opt
\gdef \listitem{\endpar
\advcount7
\startpar\hbox to \count8vu{
\hfill\ifcounteight{\marka}{\markb}{\markc}{\markd}\unskip\hbox to
25vu{}}}
\gdef \listmark{\advcount7\ifcounteight{\marka}{\markb}{\markc}{\markd}}
\gdef \par{\endpar\startpar\hbox to \count8vu{}}
}
}

% End a list
\def \endlist{
\endpar
\restorecount
\advcount8 by -1
\ifeq{\count8}0{ % If ending outermost list, reset paragraph structure and
\gdef \listitem{\listitemerror} % set error messages to be issued if an
\gdef \listmark{\listmarkerror} % attempt to specify a list item is made
\parindent \saveparindent
\gdef \par{\endpar}
}
\else{\setcount9\count9\neg9\ifpos9{\send9{Extra ENDLIST}}\else{}}
}

```



## TABLE OF CONTENTS MACROS

Lynne A. Price

BNR INC.,  
subsidiary of Bell-Northern Research, Ltd.

Automatically generated tables of contents are a convenient side effect of many computerized document preparation systems. Of course, the style of a table of contents is highly dependent on the style of chapter and section headings and whether lists of figures and tables are included. This article describes one set of TeX macros used to number chapters and sections and to produce tables of contents, lists of figures, and lists of tables. While many users will prefer other formats than the one used here, macro writers may wish to adapt this package to other conventions. The technique used involves writing one or more auxiliary files with TeX's `\send` feature as a document is formatted. These auxiliary files contain the information needed to produce a table of contents and lists of figures and tables.

The structure of a document prepared with this package is specified with the following macros:

- `\chapter {<chapter title>}` starts a new numbered chapter with the indicated heading placed at the top of the next page and entered into the table of contents. This macro automatically capitalizes all letters in its argument. In the rare situations where lowercase letters are needed, the construct

`\lowercase {<lowercase letters>}`

can be used to preserve capitalization.

- `\section {<section title>}` starts a new section with the indicated heading placed on the current page and entered into the table of contents. The heading is preceded by the chapter number and the section number separated by a period (e.g., the first section in the second chapter is labelled "2.1 Section Title"). No automatic capitalization of section titles is performed—the user should capitalize the first letter of each word. Section numbers are stored in macro `\sectioncount`.
- `\subsection {<subsection title>}` starts a new subsection with the indicated heading placed on the current page and entered into the table of contents. Subsections are labelled with the chapter number, section number, and subsection number, all separated by periods. No automatic capitalization of subsection titles is performed. Subsection numbers are stored in `\subsectioncount`.
- Some documents require more than three levels of headings. The macro `\subsub {<sub-subsection heading>}` allows for additional levels. The specified heading is placed on the current page and entered into the table of contents. However, sub-subsections are not automatically numbered. If the user wants a number to appear with the title in the text or in the table of contents, he must include it explicitly in the argument to `\subsub`.

- The macros `\figure {<figure title>}` and `\table {<table title>}` generate numbered figures and tables with the title centered in the current page. An appropriate entry is made in the list of figures or list of tables. Figures and tables are numbered relative to each chapter. Each is labelled with the chapter number and the figure or table number separated by a hyphen (*e.g.*, the first table in the second chapter is labelled 2-1). The number of the next figure to be generated is stored macro `\figurecount` and that of the next table in macro `\tablecount`.
- `\enddoc` to end the document

When processing a document in this format, TeX optionally writes one or more files of TeX input that can be processed later to generate a table of contents, list of figures, or list of tables. These auxiliary files are generated when the original input contains a call to the macro `\enablecontents`. This call must appear before the first macro that would generate a table of contents entry, *i.e.*, before the first chapter.

The default name for the table-of-contents file is CONTENTS. When the user wishes to use another file, he can specify its name with the control sequence `\contentsfile {<filename>}`. Analogously, the control sequences `\figuresfile {<filename>}` and `\tablesfile {<filename>}` govern the names of the files used for the lists of figures and tables. Defaults for the latter are FIGURES and TABLES. If any of the default names are changed, the appropriate macro calls must appear in the input before the call to `\enablecontents`. As with the names of all TeX input files, these generated files have names with the extension ".TEX". This extension is automatically supplied, whether or not the default name is used.

In order for pages to be properly numbered, the user must reserve an appropriate number of pages for the table of contents and lists of figures and tables and must, in addition, indicate where this space should be reserved. The macro `\contentshere` reserves space for the table of contents and for any lists of figures and of tables. The number of pages required for the table of contents is specified with the control sequence `\numbercontentspages {<number>}`. There are analogous control sequences `\numberfigurespages {<number>}` and `\numbertablespages {<number>}`. The default numbers of pages are 1 for the table of contents and 0 for both the list of figures and the list of tables. Since the auxiliary files for the lists of figures and tables are created only when the corresponding number of pages is nonzero, any changes to these defaults must be made before the call to `\enablecontents`. When each auxiliary file is processed, an error message is issued if the actual number of pages generated does not match the estimate.

The macros always produce a table of contents first and any list of figures before a list of tables. Two or more of the lists can be printed on one page. With the macro calls `\numberfigurespages{-1}` or `\numbertablespages{-1}`, the user indicates respectively that the list of figures or list of tables is not to start on a new page.

In general, the user must run the TeX processor separately to format the auxiliary file corresponding to each of these lists that appear in a particular paper. An exception occurs when two or more lists fit on a single page. In this case, the user must invoke TeX only for the first list on a page; lists that follow on the same page are processed automatically.

These constructs are illustrated in Figure 1, which shows a portion of the TeX input used to produce one document. The input for the two-page table of contents is stored in file `MACROCON.TEX`, that for the list of figures in `MACROFIG.TEX`, and that for the list of tables in `MACROTAB.TEX`. The lists of figures and tables are printed on the same page. The TOPS-20 command

```
◎TEX MACROCON
```

formats the table of contents while the command

```
◎TEX MACROFIG
```

was used to format both the list of figures and the list of tables. If the latter two lists did not appear on the same page, they would have been processed separately.

## Table of Contents Macros

```

\contentsfile{macrocon} % Contents go in MACROCON.TEX
\figuresfile{macrofig} % List of Figures in MACROFIG.TEX
\tablesfile{macrotab}  % List of Tables in MACROTAB.TEX
\numbercontentspages{2} % 2 pages for Table of Contents
\numberfigurespages{1} % 1 page for List of Figures
\numbertablespages{-1} % List of Tables on same page as List of Figures
\enablecontents        % Produce a table of contents

                        % Continue with rest of document
.
.
.

```

Figure 1. Preparing the Table of Contents

In rare cases, the table-of-contents file may require editing. This situation is most likely to occur when long titles are used or when macro calls occur within table-of-contents entries. Sometimes, *e.g.*, when a nonstandard font is required in a table of contents entry, the user may find it convenient to explicitly `\send` information to the table of contents or list of figures or list of tables files. These macros use character output file 1 for the table of contents, file 2 for the list of figures, and file 3 for the list of tables.

The user who is unable to recover from a TeX error received while processing a table-of-contents file should consult a TeX wizard.

The following macros provide chapter and section structure and cause the auxiliary files to be created. The auxiliary files also access these macros and assume them to be stored in a file called TUGCHAP.TEX.

```

% Chapter, figure, table, and table of contents macros
\input basic

% \neg and \ifzero from Appendix X
\def\neg#1{\setcount#1-\count#1}
\def\ifzero#1#2\else#3{\ifpos#1{#3}\else{\neg#1
  \ifpos#1{\neg#1 #3}\else{\neg#1 #2}}}}
% Arguments to \ifeq can be constants of counters
\def\ifeq#1#2#3\else#4{\setcount9 #1 \advcount9 by -#2
  \ifzero9{#3}\else{#4}}

% Macro to advance pseudo-counters (i.e., macros defined to be integers
% in order to bypass TeX's limited number of counters
\def\advcounter#1#2{\setcount9 #1\advcount9 by #2\xdef#1{\count9}}

\def\firstpage{T}
\def\chaptercount{0}
\def\figurecount{1}
\def\tablecount{1}

\def\break{\vfil\vfilneg}

% End of Document

```

```

\def\enddoc{
  \par\vfill
  % Check If Figures and Tables Lists on Same Page as TOC
  \if 0\enablecont{} \else{
    \if 0\figurespages{          % no LF
      \if 0\tablespages{}       % no LF or LT
        \else{                  % TOC, but LT
          \setcount9\tablespages
          \ifpos9{}\else{\writecon0{\Ainput \tables}} % LT on same page as TOC
        }
      }
    }
    \else{                       % LF
      \setcount9\figurespages
      \ifpos9{}
      \else{
        \if 0\figurespages{}
        \else{\writecon0{\Ainput \figures}} % LF on same page as TOC
      }
      \setcount9\tablespages
      \ifpos9{}
      \else{
        \if 0\tablespages{}
        \else{\writecon1{\Ainput \tables}} % LT on same page as LF
      }
    }
  }
}

% Close TOC files
\closecon0\contentspages{Table of Contents}
\closecon1\figurespages{List of Figures}
\closecon2\tablespages{List of Tables}
\vfill\ejct\end
}

% Chapters and sections

% \chapnum prints right-justified chapter number
\def\chapnum#1#2{\xdef\curchap{#2}\save9\hbox{#1}\save8\hbox{#2}\hskip
  1wd9\hskip-1wd8 #2}
\setcount 0 1 % Page number
\def\chapter#1{
  \if T\firstpage{}\else{\vfill\ejct} % Eject unless this is the first chapter
  \gdef\firstpage{F}
  \gdef\sectioncount{0}
  \gdef\figurecount{1}
  \gdef\tablecount{1}
  \vbox to 16ex{
    \hbox{\bf\advcounter\chaptercount1\chapnum{1.}\{\chaptercount.}\quad
      \uppercase{#1}}
    \writecon0{\vskip 2ex}
    \writecon0{\AitemincontB\bf\Achapnum{1.}\{\curchap}\quad\uppercase}
    \writecon0{\#1}\E\B\count0\#E}
    \vskip 2ex}
}

```

## Table of Contents Macros

```

\def\section#1{
  \par\vskip 4ex\break
  \gdef\subsectioncount{0}
  \advcounter\sectioncount1\hbox{\chapnum{1.1}{\chaptercount.\sectioncount}
    }\quad\it#1}
  \writeconO{\AitemincontB\save9\hbox{\bf1.\quad}\hskip 1wd9
    \Achapnum{1.1}{\curchap}\quad\B\it}
  \writeconO{\#1\T\T\B\countO\T\T}
  \penalty 1000
  \vskip 1ex
}

\def\subsection#1{
  \par
  \vskip 4ex
  \break
  \advcounter\subsectioncount1
  \hbox{\chapnum{1.1.1}{\chaptercount.\sectioncount.\subsectioncount}\quad
    \it#1}
  \writeconO{\AitemincontB\save9\hbox{\bf1.\quad\rm1.1\quad}\hskip
    1wd9\Achapnum{1.1.1}{\curchap}\quad\B\it}
  \writeconO{\#1\T\T\B\countO\T\T}
  \penalty 1000
  \vskip 1ex
}

\def\subsub#1{
  \par
  \vskip 4ex
  \break
  \hbox{\it#1}
  \writeconO{\AitemincontB\save9\hbox{\bf1.\quad\rm1.1\quad1.1.1\quad}\hskip
    1wd9\B\it}
  \writeconO{\#1\T\T\B\countO\T\T}
  \penalty 1000
  \vskip 1ex
}

% Table of Contents
% Note: to avoid immediate evaluation of some macros, and to be able to
% write lines with unbalanced grouping characters to the table of contents
% files, CTRL-A is used as an escape character (\) and CTRL-B and CTRL-E are
% used as grouping characters (left and right braces, respectively).

\def\writecon#1#2{\if T\enablecon {\send#1{#2}} \else {}}

% macros for defining file names and estimated page counts
\def\contentsfile#1{\gdef\contents{#1}}
\def\figuresfile#1{\gdef\figures{#1}}
\def\tablesfile#1{\gdef\tables{#1}}
\def\numbercontentspages#1{\gdef\contentspages{#1}}
\def\numberfigurespages#1{\gdef\figurespages{#1}}
\def\numbertablespages#1{\gdef\tablespages{#1}}

```

```

% set defaults
\contentsfile{CONTENTS}
\figuresfile{FIGURES}
\tablesfile{TABLES}
\numbercontentspages{1}
\numberfigurespages{0}
\numbertablespages{0}

% Numcon writes on contents files information needed to determine the
% page number on which the contents or list of figures or tables should
% start. It also makes table of contents entries for the contents,
% and lists of figures or tables. The first parameter is 0, 1, or 2
% to indicate which list is involved; the second is the number of
% pages that list is supposed to take; the third is the title of the list.
\def\numcon#1#2#3{
  \if 0#2{
  \else{
    \writecon0{\fAstartpage
      {\count0}{\contentspages}{\figurespages}{\tablespages}{#1}}
    \setcount9 #2
    \ifpos9{
      \writecon#1{\fAstartpage
        {\count0}{\contentspages}{\figurespages}{\tablespages}{#1}}
      \writecon#1{\fAsetcount0 \fAccount9}
      \writecon#1{\fAxdef\fAsavestart{\fAstart}}
      \advcount0 by #2
    }
  \else{
    \writecon0{\vskip 2ex}
    \writecon0{\fAitemincon{\bf #3}}{\fAstart}}
  }
}

\def\contentshere{
  \if T\firstpage{\else{\vfill\object} % Eject unless this is the first page
  \numcon0{\contentspages}{CONTENTS}
  \numcon1{\figurespages}{LIST OF FIGURES}
  \numcon2{\tablespages}{LIST OF TABLES}
  \def\firstpage{T}
}

% Initialize a particular table of contents file (called from \enablecontents)
% Parameter 1 indicates which list, parameter 2 is title of list,
% parameter 3 is estimated number of pages, parameter 4 is file name on
% which list is to be written
\def\initcon#1#2#3#4{
  \if 0#3{
  \else{
    \if T\enablecon {\open#1=#4.TEX } \else {}
    \setcount9 #3
    \ifpos9{
      \writecon#1{\input tugchap}

```

## Table of Contents Macros

```

    \writecon#1{\chcode`001_0}
    \writecon#1{\chcode`002_1}
    \writecon#1{\chcode`005_2}
    \writecon#1{\input tugtoc}
    \writecon#1{\Adef\Atarget{#3}}
  }
  \else{}
  \writecon#1{\Aheading\B#2\E}
}
}

\def\enablecon{F} % flag indicates whether table of contents files are enabled
\def\enablecontents{
  \gdef\enablecon{T}
  \initcon0{CONTENTS}\contentspages\contents
  \initcon1{List of Figures}\figurespages\figures
  \initcon2{List of Tables}\tablespages\tables
}

% Close a table of contents file, write code to check whether estimated
% number of pages was correct
\def\closecon#1#2#3{
  \if 0#2{}
  \else {
    \setcount9 #2
    \ifpos9{
      \writecon#1{\vfill\eject}
      \writecon#1{\Asetcount9 \Asavestart}
      \writecon#1{\Aadvcount 9 by \Atarget}
      \writecon#1{\Aadvcount 9 by -\Account0}
      \writecon#1{\Aadvcount 9 by -1}
      \writecon#1{\Aifzero 9 {} \Aelse{\Asend9{#3 Page Estimate Incorrect}}}
      \writecon#1{\end}
    }
    \else{}
  }
}

% Figures and Tables

\def\figtab#1#2#3#4{
  \vskip 3ex
  \xdef\tabnum{\chaptercount-#3}
  \vbox{\ctrline{#4 \tabnum\quad #2}
  \if T\enablecon {
    \writecon#1{\Aitemincont\B\Achapnum{i-1}{\tabnum}\quad#2\E\B\count0\E}}
  \else {}
  }
  \advcounter{#3}1
  \vskip 3ex
}

```

```
\def\figure#1{\figtab1{#1}\figurecount{Figure}}
\def\table#1{\figtab2{#1}\tablecount{Table}}
```

The following macros, assumed to be stored in a file called TUGTOC.TEX are required by the auxiliary files.

```
% Table of contents macros
\def\lead{\leaders\hbox to 8pt{\hfill}\hfill}
\def\heading#1{\vbox to 16ex{}\ctrlline{\bf #1}\vskip 2ex}

% Advance counter 8 by value in counter 9 (used in calculating starting
% page number
\def\adveight#1{\setcount9 #1 \ifpos9{\advcount8 by -\count9}\else{}}

\def\startpage#1#2#3#4#5{ % Compute page # where current list starts
  \setcount8 #1 % # of first page after TOC, LF, LI
  \adveight{#4} % - # of pages for LI
  \if 2#5{\setcount 7 #4 % doing LI, check if not on new page
    \neg7\ifpos7{\advcount8 by -1}\else{}}
  \else{\adveight{#3}} % - # pages for LF unless doing LI
  \if 0#5{\adveight{#2}} \else{} % - # pages for TOC if doing TOC
  \if 1#5{\setcount 7 #3 % doing LF, check if not on new page
    \neg7\ifpos7{\advcount8 by -1}\else{}} \else{}
  \xdef\start{\count8}
  \advcount 8 by -1
}

\def\itemincon#1#2{\hbox to size{#1}\lead\hbox to 1wd0{\setcount9 #2
  \advcount9 by -1\hfill\count9}}

\save0\hbox{1000} % Set box 0 to a very large page number to determine
% maximum width
```



## Utility Macros

## UTILITY MACROS

Patrick Milligan  
Lynne A. Price

BNR INC.,  
subsidiary of Bell-Northern Research, Ltd.

As a part of our experience with the creation and use of T<sub>E</sub>X macros, several small but useful macros have been written to aid in the creation of large macro packages. This article highlights these utility macros, and their usage.

*Font Definition*

In order to facilitate the definition and use of fonts not declared in BASIC.TEX, a macro called `\fontdef` was written to declare a font and define a macro to invoke it. This macro takes three arguments:

```
\fontdef {<Font Code>}{<Font Name>}{<Macro Name>}
```

For example, the standard definition of `\rm` from BASIC.TEX would look like:

```
\fontdef {a}{cmr10}{\rm}
```

In addition, we have adopted the convention that our standard macro packages never use uppercase letters as Font Codes, so that users always know these letters can be used for any special fonts declared in a specific document.

The source for `\fontdef` follows:

```
% The macro fontdef is used to declare fonts and define a macro
% that invokes them.
\def\fontdef#1#2#3{\font #1=#2 \def #3{\curfont #1}}
```

*Counter Value Comparison*

To extend the macros `\neg` and `\ifzero` given in Appendix X of the T<sub>E</sub>X manual, we have created a macro called `\ifeq` which tests equality between two values (which can either be constants or counters). `\ifeq` takes four arguments:

```
\ifeq {<Value 1>}{<Value 2>}{<Then Clause>}\else{<Else Clause>}
```

This macro uses counter 9 as a scratch counter. We have found that always having a scratch counter available is a reasonable way to implement general counter arithmetic. The following example of `\ifeq` compares counter 0 equal to 1:

```
\ifeq {\count0}{1}{Is one}\else{Isn't one}
```

The source for `\ifeq` follows:

```

% The macros \neg and \ifzero are copied from Appendix X.
% We have added \ifeq. Unlike other similar macros, \ifeq expects
% its arguments to be values, so if a counter is used it must be
% specified (e.g., \count3 instead of 3) and constants are permitted.
\def\neg#1{\setcount#1-\count#1}
\def\ifzero#1#2\else#3{\ifpos#1{#3}\else{\neg#1
  \ifpos#1{\neg#1 #3}\else{\neg#1 #2}}
\def\ifeq#1#2#3\else#4{\setcount9 #1 \advcount9 by -#2
  \ifzero9{#3}\else{#4}}

```

### *Pseudo Counters*

In writing large macro packages which keep track of page, chapter, section, subsection, table, and figure numbers, it is likely that the ten counters provided by TeX will not be adequate. One alternative is to use macros to hold counter values. By using `\xdef` and `\setcount`, it is possible to convert counters into macros, and *vice versa*. To facilitate the advancing of pseudo counters, a macro called `\advcounter` was written. This macro takes two arguments:

```
\advcounter {<Pseudo Counter>}{<Advance Value>}
```

Counter 9 is used as a scratch counter in `\advcounter`. The following example “sets” counter `\pagenum` to 1, then “advances” it by 2:

```
\def \pagenum{1} \advcounter \pagenum {2}
```

The source for `\advcounter` follows:

```

% Macro to advance pseudo-counters (i.e., macros defined to be integers
% in order to bypass TeX's limited number of counters)
\def\advcounter#1#2{\setcount9 #1\advcount9 by #2\xdef#1{\count9}}

```

### *Uppercase Roman Numerals*

TeX's facility for lowercase Roman numerals is useful in a variety of applications. However, it is not obvious how to obtain uppercase Roman numerals! Assuming that counter 0 holds a negative value the intuitive attempt

```
\uppercase{\count0}
```

doesn't work since `\uppercase` see `\count0` as a single, unexpanded token, not as a token list consisting of the Roman numeral equivalent. By using `\xdef`, we can force the expansion of the negative counter to the Roman numeral string, allowing `\uppercase` to produce the desired uppercase Roman numeral. Thus, the (non-obvious) sequence

```
\xdef \num{\uppercase{\count0}} \num
```

is what we want!

We have written two macros to return the upper or lower case Roman equivalent of a positive counter. The macro `\roman` returns a lowercase Roman numeral, and the macro `\Roman` returns an uppercase Roman numeral. Both of these macros use counter 9 as a scratch counter.

The source for these macros follows:

Utility Macros

```
% Provide for converting positive counters to upper or lower case Roman
\def\roman#1{\setcount@ -\count#1\count@}
\def\Roman#1{\setcount@
-\count#1\edef\uppercaseroman{\uppercase{\count@}}\uppercaseroman}
```

הצעת חוק

הצעת חוק להקמת ועדה לחקירת מעורבות ישראלים במשפט השואה.  
 מטרת החוק היא להקמת ועדה שתחקיר את מעורבותם של ישראלים במשפט השואה.  
 הוועדה תהיה בראשות שופט בדימוס ויהיו לה שני חברים נוספים.  
 תפקידה של הוועדה יהיה לחקור את מעורבותם של ישראלים במשפט השואה.  
 הוועדה תהיה רשאית להקדם את חקירתה בכל אופן שייראה לה.  
 הוועדה תהיה רשאית להזמין את כל מי שיש לו מידע רלוונטי לחקירה.  
 הוועדה תהיה רשאית להעביר את ממצאי חקירתה לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.  
 הוועדה תהיה רשאית להגיש דו"ח לרשות המוסמכת.

```

\input letter
\fontdef H{hebrew}{\hebrew}
\def\threedots{\downer{\ldots}}
\def\colon{\downer{:}}
\def\tex{\downer{\rm\TeX}}
\def\leftparen{\downer{(}}
\def\rightparen{\downer{)}}
\def\bang{\downer{!}}
\def\downer#1{\lower 5pt\hbox{#1}}
\hebrew
\hbox{\downer{1980} RBMBVN}
\rjustline{\bang BR mVLS RYMA JSYLAL}
\hbox to size{XVWNH mJ HNRMGW HLAH WVR&H LKS HVQNV HRYHM HMLXH kL
LXAL XRKVM YNA WYSAR\hskip 2em}
\hbox to size{VYHY ALS mJP DVJ LXAM YNA nKLV mYLVX YWBL WVNLCB kL mG
nYA YL VMKS HARNK .YXKVNH}
\rjustline{.mYPCVN mYYSQ}

\hbox to size{DYWJBS HVQM YNA .ABAL WNNVLWH HZ LJS nYBM YNAV
.YWBWK ALS BR nmZ HZ\hskip 2em}
\rjustline{.WVBVSW LBQY kBWKM LS RWVY LVDG ZVXA}

\hbox to size{WYRBJB CYPDHL LQ kK LK AL .YLS HDVBJB YNVRTQLAH BSXMH
YDY LJ CPDVM HZH BWKMH\hskip 2em}
\hbox to size{HZH CVPDHS HA&MHH BGA kRD \threedots kVPH RDCE WYLGNAH
WVYVAH WA CYPDHL mYKYR& YRHS}
\hbox to size{WAV mYRSYH mYVQH WA WYTMWM HRV&B mYRAWM \colon CVPD
WVYVA YNYM LK RV&Y WRSPAM HYLJ CCVBM}
\hbox to size{mYRMVAS ALA SY .CVPD nVNGC WVVAB WYVVAH LK WA R&YM BSXMHV
mY&VXNH WWSQH}
\rjustline{.GRBNTVG nMZM RWVYB HBVSXH CVPD WA&MH AYH {\tex}
WARQNH WAZH HA&MHHS}

\hbox to size{SDXH WYBH nYB QXRMH .HRYD VNRBJ WJMSS YPK .WRQBS ZAM
Y&XV HNS HZ\hskip 2em}
\hbox to size{WYBH XTS kA LDVG WVVAB kRJB AYH HSDXH R&XH
.mYRIMVLQ JBRAM WVXP AVH nSYHV}
\hbox to size{HBYCHV \leftparen LBX HZV\rightparen\ mDVQH WYBB VMK
YRP Y&J VA QRY WYNYG VNL nYA .mYNS YP LVDG RWVY}
\hbox to size{AVH RWVYB LVDGH nVLAH LS HBVGH \colon mYYQNJ nVLA
Y&J HSVLS mYLDG R&XBS AYH}
\rjustline{.mYKRS LDGL LYXWA YLVA kA .RTM YNSK VJZG RTVQV
RTM \downer{25}K}

```

```

\hbox to size{.JVBS DVJB VNWYBB mYNWXWM HVANV nRVA kL JVDYK .HMLSV
  HAYRB AYH nAK HXPSMH\hskip 2em}
\hbox to size{.mYNVRTQLA mYBSXMB CRVQ HLYXWM YCNN RAVNAYB .XMS VYHY
  YADVBV mYXRVA \downer{60}K VNL VYHY}
\hbox to size{.HZ JV&QMB DVEJL kK RXA HVQMV .mYBSXML WVYNKVV WBYWK
  mYNS \downer{3-2} kSMB DVMLL HVQM AYH}
\rjustline{.JVBSB mYMJP SVLS nVNGL mYKLVH mYLVDG mYBBVS mH LAKYMV DVD}

\hbox to size{WVKYA HNSH HYNRVPYLQB .\downer{\rm Zinfandel} YNSV
  \downer{\rm Cabernet} DXA nYY YNM YNS YWYSJ HNSH\hskip 2em}
\hbox to size{HMLS LS VWJD HWYH HM BGA kRD \threedots\ mYNS mYRSJ
  ZAM RfVYB HBVTH HWYH mYBNJH}
\rjustline{.QMjH RMSMM VNLBQS BVTH SBDH DJB HMJNLV VL HDVV .YwXlSS
  nYYL RSQB}

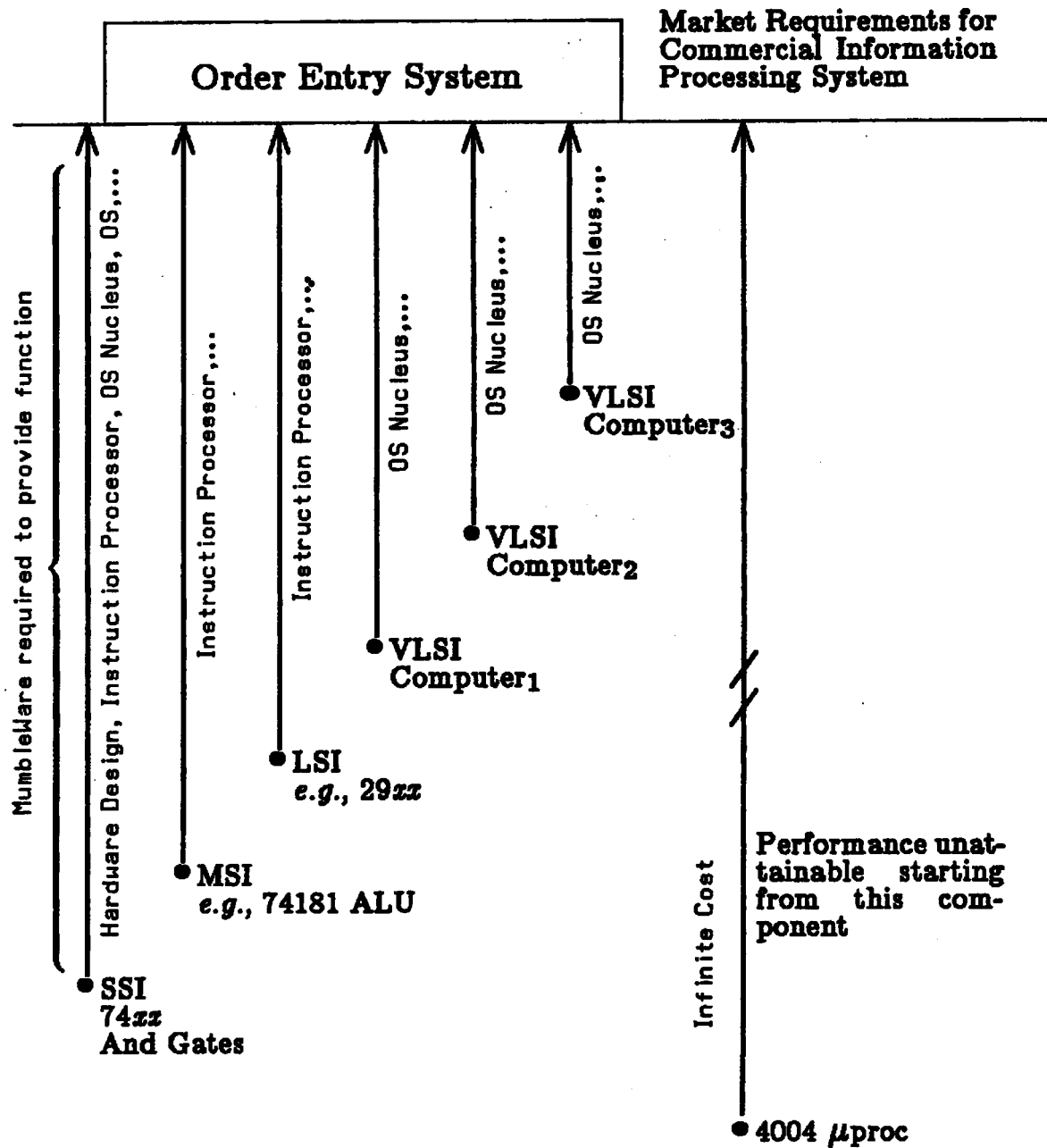
\vskip 2ex
\ctrline{.mLVKL S$\prime\prime$DV WVARWHL\hskip 1in}
\vskip 6ex
\hbox{nYBVR DVD YMR}
\endletter

```

Technology

October 1980

### Alternative Development Strategies



TEX

Sample



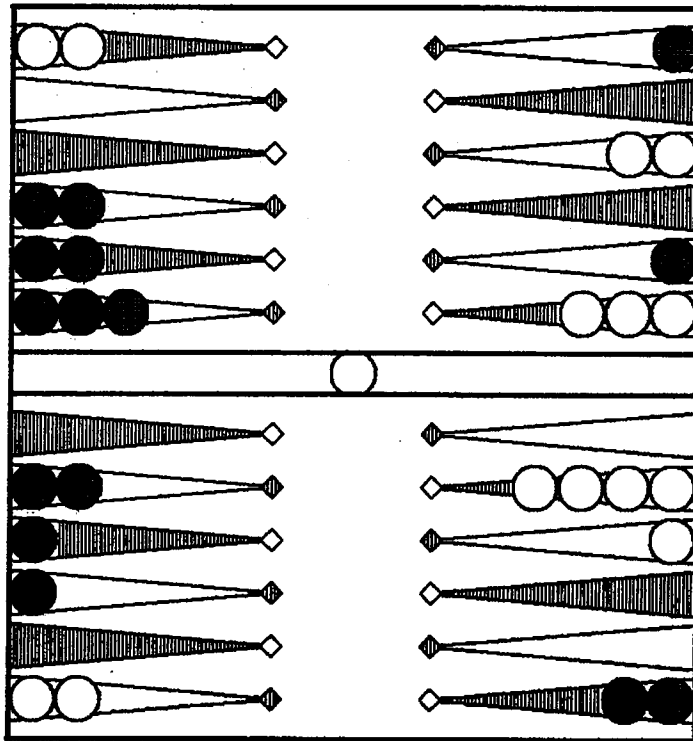
Дерево плаваєт в водѣ.



שלום



$$\int_{-\infty}^{\infty} \sin 2\theta d\theta$$



Ἄλεκτρον φέλη ὅτι μοι



BNR INC.



\* \* \* \* \*

## Letters

\* \* \* \* \*

## REPORT FROM AN EARLY AMS-TEX USER

David Eck

I was very happy recently to bring to a successful conclusion my first major project in using TeX—actually, my first experience of any kind with TeX. The project was the preparation of my thesis for publication in the *Memoirs of the American Mathematical Society*, which involved the production of about fifty typeset pages, with a rather high density of complex mathematical symbols.

Having typed one or two mathematical papers myself and read my share of typewritten mathematics, I was impressed with both the ease with which input for TeX can be prepared and with the quality of the output. Of course, this praise is not without qualification, but I believe that a large fraction of the frustrations I encountered can be traced to the “frontier” nature of the conditions under which I was working.

I began with a quick reading of Knuth’s manual. I then spent a good bit of time, interrupted by frequent references to the manual, transcribing the first few pages of my thesis into a TeX input file. I was surprised after a little more than ten pages to find that I was typing at almost my (admittedly slow) normal typing speed, even without pre-processing the manuscript in any way. About a third of the way through, I received a rudimentary version of the forthcoming AMS-TeX manual, which made things a bit easier. Since the book had no index at the time, it was rather difficult to use, but it is clear that AMS-TeX will insulate the user from many of the traps and technicalities of TeX. The average time for typing a page, after I overcame my initial inexperience, turned out to be about one-half hour, and the task seemed less unpleasant than ordinary mathematical typing.

I did not try to run anything in TeX until the entire thesis was prepared. This proved to be a major mistake. I had to spend perhaps an additional forty-five minutes per page at a computer terminal correcting and editing my files. More than half of this could certainly have been avoided if I could have seen my output every few pages, and learned from my mistakes. Unfortunately, this was not possible, since I was at Dartmouth College, and the nearest TeX program was at MIT. Hopefully, this sort of thing will not remain a problem for long!

My overall impressions of TeX were very favorable. I wish it were available even for day-to-day things such as preparing tests. I am especially fond of the ease with which errors can be corrected and passages added or deleted in a computer file. Let us all look forward to the time when TeX becomes a *lingua franca* of mathematical publication. My experience has convinced me that this would make life easier for the authors and typists, as well as for the publishers.

Editor’s note: David Eck’s manuscript has been run successfully through TeX and a prototype AMS-TeX both at MIT and at the AMS, and output onto various devices at those locations (and elsewhere). Appendix A contains several one-page samples (along with the TeX input) to allow a comparison of the appearance and quality available from both low and high resolution devices.

Eck’s paper was written before the specifications were set up for the *Memoirs*. A header (macro) package is being written, however, and should be available soon for distribution with AMS-TeX. The sample input should thus not be taken as the archetype of *Memoirs* input.

\* \* \* \* \*

Dear Editor,

I was pleased to receive the first issue of TUGboat, and am looking forward to the day (any year now, they tell me) when TeX will be available on our system at Brown.

However, I was disturbed by one thing in TUGboat: the large number of puns predicated on the assumption that TeX rhymes with sex. The title of Michael Spivak’s book will further popularize this myth. I would have thought that the high wizards associated with TUGboat would have had occasion to actually read Don Knuth’s manual; if they had looked at page 4, they would know that TeX, Tau Epsilon Chi, rhymes with bleeehhh. As Knuth says, you can pronounce it however you want and the computer won’t mind. To us humans, however, there is a useful difference: If one is looking for advice on TeX, one should shun anyone claiming to be a TeXpert; they probably know a lot less than they think they do. On the other hand, a TeXnician is a most useful friend to have.

Sincerely,  
Graeme Hirst  
Department of Computer Science  
Brown University

3 February 1981

## APPENDIX A

### Output Samples from a Paper by David Eck

The pages given here are various renditions of one page from a paper by David Eck of Dartmouth College, a first-time user of  $\text{T}_{\text{E}}\text{X}$  and  $\text{A}_{\text{M}}\text{S}-\text{T}_{\text{E}}\text{X}$ . (He describes his experience on page 127.) One page was selected from his paper, and run through  $\text{T}_{\text{E}}\text{X}$  at three locations: the American Mathematical Society, Massachusetts Institute of Technology, and Stanford. The composed file was then output to whatever devices were available at each location. The samples are given in the following order.

- input file listing;
- Florida Data Model BNY dot matrix printer, 128 dots/inch;
- Xerox XGP, 200 dots/inch;
- Benson-Varian 9211, 200 dots/inch;
- Canon Laser Beam Printer, 240 dots/inch;
- Xerox Dover, 384 dots/inch;
- Alphatype CRS, 5300 dots/inch.

At the AMS a few rudimentary timing estimates were made. For such a small sample, these times probably represent an upper bound, rather than a true measure of time required to process this type of material. The computer involved is a DEC 2060 with 512K words of memory. Output to the Varian is controlled by a spooler, which removes the processing from user control once a job has been released to the queue. No times were obtained for output to any device but the Varian, but an attempt will be made to obtain and publish such information in a later issue of TUGboat.

- $\text{T}_{\text{E}}\text{X}$ : CPU time of  $< 6$  sec. included loading the program and reading all font and macro header files; net time required for processing the one page of input was probably  $< 3$  sec.
- release to Varian spooler: 1 CPU sec.
- printing on Varian:  $< 4$  CPU sec.

Output from another device, a Compugraphic 8600, is shown and described in the article by Ralph Stromquist, page 51. The macro packages and other items by Lynne Price and Pat Milligan (pages 87-126) were output on a Versatec, which is essentially similar to the Varian 9211. The two articles by Lawson, Zabala and Díaz (pages 20-47) were output on a Dover, as were the macro package descriptions by Max Díaz and Arthur Keller. The body of this newsletter was output on the Varian at the AMS.

\*This is eck secl

\chapterbegin{\section 1: The Functors  $(\cdot)^n_k$ }

\definition Definition 1.1\\ If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)^n_k$  as follows:  
 If  $M$  is a  $C^\infty$  manifold, then  $(\cdot)^n_k M = \left\{ \varphi : \mathbb{R}^n \rightarrow M \mid \varphi(0) = p \right\}$  where  $\varphi(0)$  is the  $k$ -jet of  $\varphi$  at  $0$ . If  $f: M \rightarrow N$  is a smooth map, then  $(\cdot)^n_k f: (\cdot)^n_k M \rightarrow (\cdot)^n_k N$  is defined by  $(\cdot)^n_k f(\varphi) = \varphi \circ f$ .

Note that  $(\cdot)^n_k M$  is a fiber bundle over  $M$ , with projection  $\pi: (\cdot)^n_k M \rightarrow M$  given by  $\pi(\varphi) = \varphi(0)$ . In particular,  $(\cdot)^1_1 M$  is just the tangent bundle  $T M$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  which agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)^n_k$ . They are easy to establish, and we omit the proofs.

\theorem Theorem 1.2\\ a) If  $M$  and  $N$  are manifolds, then  $(\cdot)^n_k M \times (\cdot)^n_k N$  and  $(\cdot)^n_k (M \times N)$  are naturally equivalent.  
 b) If  $M$  is a manifold, then  $(\cdot)^m_k M$  and  $(\cdot)^{m+k}_k M$  are naturally equivalent. \QED

We note that by naturality here, we mean, in a), that given any maps  $f: M \rightarrow N$ ,  $g: N \rightarrow P$ , the diagram  $(\cdot)^n_k M \times (\cdot)^n_k N \xrightarrow{f \times g} (\cdot)^n_k (M \times N) \xrightarrow{g} (\cdot)^n_k P$  commutes, and similarly for b).

\par It may be useful to see what the map  $\Psi: (\cdot)^m_k M \rightarrow (\cdot)^{m+k}_k M$  looks like in coordinates:

If  $y_1, \dots, y_s$  are local coordinates on  $M$ , we get local coordinates  $y^\alpha$ ,  $\alpha = (1, \dots, n)$  with  $|\alpha| \leq k$ , on  $(\cdot)^n_k M$  by  $y^\alpha(\varphi) = \frac{1}{k!} \partial^\alpha x(y \circ \varphi)(0)$ . By extension, we have coordinates  $y^{\alpha; \beta}$  on  $(\cdot)^m_k M$  and  $\overline{y^{\beta; \alpha}}$  on  $(\cdot)^{m+k}_k M$ , where  $\beta = (1, \dots, m)$  with  $|\beta| \leq k$ . The map  $\Psi$  is given in these coordinates by  $\Psi(\varphi) = \overline{y^{\beta; \alpha}}(\varphi) = y^{\alpha; \beta}(\varphi)$ .

We will use the following notation:  $\mathcal{M}^n$  will denote the set of all smooth maps  $\mathbb{R}^n \rightarrow M$ . Whenever we consider  $\mathcal{M}^n$  as a topological space, we will always use the  $C^\infty$  topology. If  $p \in M$ , then we denote the constant map  $\mathbb{R}^n \rightarrow M$  which sends each element of  $\mathbb{R}^n$  to  $p$  by  $A_p$ .

## Florida Data Model BNY

### §1: THE FUNCTORS $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{ j_k(\varphi)_0 \mid \varphi: \mathbb{R}^n \rightarrow M \}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f: M \rightarrow N$  is a smooth map, then  $f_k^n: M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi: M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  which agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_l^m$  and  $(M_l^m)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f: M \rightarrow M'$  and  $g: N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M'_k^n \times N'_k^n) \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\psi: (M_k^n)_l^m \rightarrow (M_l^m)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_n$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$

§1: THE FUNCTORS  $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{ j_k(\varphi)_0 \mid \varphi : \mathbb{R}^n \rightarrow M \}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f : M \rightarrow N$  is a smooth map, then  $f_k^n : M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi : M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  that agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_\ell^m$  and  $(M_\ell^m)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f : M \rightarrow M'$  and  $g : N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M'_k^n \times N'_k^n) \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\Psi : (M_k^n)_\ell^m \rightarrow (M_\ell^m)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_n$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$

By extension, we have coordinates  $y_i^{\alpha;\beta}$  on  $(M_k^n)_\ell^m$  and  $\bar{y}_i^{\beta;\alpha}$  on  $(M_\ell^m)_k^n$ , where  $\beta = (\beta_1, \dots, \beta_m)$  with  $|\beta| \leq \ell$ . The map  $\Psi$  is given in these coordinates by  $\Psi((y^{\alpha;\beta})) = (\bar{y}^{\beta;\alpha})$  where  $\bar{y}_i^{\beta;\alpha} = y_i^{\alpha;\beta}$ .

§1: THE FUNCTORS  $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{ j_k(\varphi)_0 \mid \varphi: \mathbb{R}^n \rightarrow M \}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f: M \rightarrow N$  is a smooth map, then  $f_k^n: M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi: M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  which agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_\ell^m$  and  $(M_\ell^m)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f: M \rightarrow M'$  and  $g: N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M')_k^n \times (N')_k^n \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\Psi: (M_k^n)_\ell^m \rightarrow (M_\ell^m)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_s$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$

§1: THE FUNCTORS  $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{ j_k(\varphi)_0 \mid \varphi: \mathbb{R}^n \rightarrow M \}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f: M \rightarrow N$  is a smooth map, then  $f_k^n: M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi: M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  that agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_\ell^m$  and  $(M_\ell^m)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f: M \rightarrow M'$  and  $g: N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M')_k^n \times (N')_k^n \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\Psi: (M_k^n)_\ell^m \rightarrow (M_\ell^m)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_s$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$

§1: THE FUNCTORS  $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{ j_k(\varphi)_0 \mid \varphi : \mathbb{R}^n \rightarrow M \}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f : M \rightarrow N$  is a smooth map, then  $f_k^n : M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi : M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  that agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_k^n$  and  $(M_k^n)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f : M \rightarrow M'$  and  $g : N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M')_k^n \times (N')_k^n \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\Psi : (M_k^n)_k^n \rightarrow (M_k^n)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_s$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$



# Alphatype CRS

## §1: THE FUNCTORS $(\cdot)_k^n$

DEFINITION 1.1: If  $n \geq 0$  and  $k \geq 0$  are integers, we define a functor  $(\cdot)_k^n$  as follows: If  $M$  is a  $C^\infty$  manifold, then

$$M_k^n = \{j_k(\varphi)_0 \mid \varphi: \mathbb{R}^n \rightarrow M\}$$

where  $j_k(\varphi)_0$  is the  $k$ -jet of  $\varphi$  at 0. If  $f: M \rightarrow N$  is a smooth map, then  $f_k^n: M_k^n \rightarrow N_k^n$  is defined by

$$f_k^n(j_k(\varphi)_0) = j_k(f \circ \varphi)_0.$$

Note that  $M_k^n$  is a fiber bundle over  $M$ , with projection  $\pi: M_k^n \rightarrow M$  given by  $\pi(j_k(\varphi)_0) = \varphi(0)$ . In particular,  $M_1^1$  is just the tangent bundle  $TM$ . This is clear if we consider a tangent vector at a point  $p$  of  $M$  to be an equivalence class of curves in  $M$  which agree up to the first order at  $p$ . We will need to know the following basic properties of the functor  $(\cdot)_k^n$ . They are easy to establish, and we omit the proofs.

THEOREM 1.2: a) If  $M$  and  $N$  are manifolds, then  $M_k^n \times N_k^n$  and  $(M \times N)_k^n$  are naturally equivalent.

b) If  $M$  is a manifold, then  $(M_k^n)_l^m$  and  $(M_l^m)_k^n$  are naturally equivalent. ■

We note that by naturality here, we mean, in a), that given any maps  $f: M \rightarrow M'$  and  $g: N \rightarrow N'$ , the diagram

$$\begin{array}{ccc} (M \times N)_k^n & \xrightarrow{(f \times g)_k^n} & (M' \times N')_k^n \\ \downarrow & & \downarrow \\ (M_k^n \times N_k^n) & \xrightarrow{f_k^n \times g_k^n} & (M')_k^n \times (N')_k^n \end{array}$$

commutes, and similarly for b).

It may be useful to see what the map  $\Psi: (M_k^n)_l^m \rightarrow (M_l^m)_k^n$  looks like in coordinates:

If  $y_1, \dots, y_s$  are local coordinates on  $M$ , we get local coordinates  $y_i^\alpha$ ,  $\alpha = (\alpha_1, \dots, \alpha_n)$  with  $|\alpha| \leq k$ , on  $M_k^n$  by

$$y_i^\alpha(j_k(\varphi)_0) = \frac{\partial^\alpha y_i \circ \varphi}{\partial x^\alpha}(0).$$

## Contents — Continued

### Warnings & Limitations

Barbara Beeton. *Disappearing Digits; Undisciplined Uppercase* . . . . . 53

### Macros

Barbara Beeton. *How to Prepare a File for Publication in TUGboat* . . . . . 53

Max Díaz. *T<sub>E</sub>X Macro Package* . . . . . 55

Arthur M. Keller. *Anatomy of a T<sub>E</sub>X Macro Package* . . . . . 56

Lynne A. Price and Patrick Milligan. *NOFILL Program with Pascal Source* . . . . . 87

Lynne A. Price. *List Macros* . . . . . 98

Lynne A. Price. *Table of Contents Macros* . . . . . 111

Patrick Milligan and Lynne A. Price. *Utility Macros* . . . . . 119

Lynne A. Price. *Hebrew Letter (with Source)* . . . . . 122

Lynne A. Price. *Two Slides* . . . . . 125

### Letters

David Eck. *Report from an Early AMS-T<sub>E</sub>X User* . . . . . 127

Graeme Hirst. . . . . 127

### Miscellaneous

Appendix A. *Output Samples from a Paper by David Eck* . . . . . 128



### T<sub>E</sub>X Errata

### TUG Membership List

Registration Form for Implementors' Workshop, May 14–15

Transmittal Form for TUGboat Articles on Magnetic Tape

Membership Application


 TUGboat needs your contributions. Letters and articles are welcome on any subject related to T<sub>E</sub>X and its uses. Your experiences as a T<sub>E</sub>X user may encourage or benefit some other user. Especially welcome are technical details concerning T<sub>E</sub>X installation on various architectures, output devices, and their interfaces. A special appeal has been made in this issue for letters giving your views on how T<sub>E</sub>X is to be supported in the future; see the articles by Bob Morris and Sam Whidden on pages 7–10.
 

Articles may be submitted in the form of T<sub>E</sub>X input on magnetic tape. A form enclosed with this issue gives details.



# WORKSHOP



# ANNOUNCEMENT

**T<sub>E</sub>X Implementors' Workshop**

**Thursday-Friday, May 14-15, 1981**

**Stanford University, Stanford, California**

**A Workshop for present and prospective implementors of T<sub>E</sub>X will be held at Stanford under the sponsorship of TUG. The first day, Thursday, May 14, will be devoted to technical presentations. Topics for discussion include:**

**Implementations of T<sub>E</sub>X  
Transportation of T<sub>E</sub>X among systems  
Output devices**

**On the second day, Friday, May 15, informal sessions for architecture groups will be arranged. There will also be demonstrations of T<sub>E</sub>X for new users by Don Knuth and others of the Stanford group.**

**A fee will be charged for attendance at this Workshop, with proceeds going to TUG. A form is enclosed, giving details of registration.**

**Program details will be available within a few weeks. To register, return the enclosed form to the T<sub>E</sub>X Users Group, c/o American Mathematical Society, P.O. Box 6248, Providence, R.I. 02940.**

**Participants should make their own arrangements for accommodations. A block of rooms has been reserved at the Palo Alto Holiday Inn, 625 El Camino Real, Palo Alto, Calif. 94305, (415) 328-2800. When requesting reservations, refer to the T<sub>E</sub>X Users Group; call before April 29. (An effort will be made to locate low-cost hotel accommodations as well; further information may be available later.)**