Rather listen to the sad story of mankind, who like children lived until I gave them understanding and a portion of reason. ... Numbers I invented for them, the chiefest of all discoveries; I taught them the grouping of letters, to be a memorial and record of the past, the mistress of the arts and mother of the Muses.

Aeschylus
Prometheus Bound,
translated by Paul Elmer More,
Complete Greek Drama,
Random House, 1938

TUGBOAT

THE TEX USERS GROUP NEWSLETTER
EDITOR ROBERT WELLAND

VOLUME 2, NUMBER 3 NOVEMBER 1981
PROVIDENCE RHODE ISLAND U.S.A.

ADDRESSES OF OFFICERS, AUTHORS AND OTHERS

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

BLAIR, John
CALMA
Research and Development
212 Gibraltar Drive
Sunnyvale, CA 94086
408-744-1950

CARNES, Lance
163 Linden Lane
Mill Valley, CA 94941
415-388-8853

COLE, J. M.
17 St Mary's Mount
Leyburn
North Yorkshire DL8 5JB, England

CRAIG, Jerry
Morgantown Energy Technology Center
B1-330
Collins Ferry Road
Morgantown, WV 26506
304-599-7178

DÍAZ, Max
IIMAS-UNAM
Apartado 20-726
Mexico, (20, D.F.) Mexico

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

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

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

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

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

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

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

JACKSON, Calvin W.
Computer Science Department
California Inst of Tech
256-80
Pasadena, CA 91125
213-356-6245

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

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

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

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

MOONEY, James D.
Department of Statistics and Computer Science
West Virginia University
Morgantown, WV 26506
304-293-3607

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

MURPHY, Timothy
University of Dublin
School of Mathematics
39 Trinity College
Dublin 2, Ireland
Dublin 772941, ext. 1983

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

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

PIERCE, Tom
Rohm & Haas Research Labs
727 Norristown Road
Spring House, PA 19477
215-641-7875

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

PLASS, Michael F.
Imaging Sciences Laboratory
Xerox Corporation
3333 Coyote Hill Road
Palo Alto, CA 94304
415-494-4810

PLASS, Susan
Polya 203
Center for Information Technology
Stanford University
Stanford, CA 94305
415-497-1322

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

RODGERS, David L.
c/o University of Michigan Computing Center
1075 Beal Avenue
Ann Arbor, MI 48109

ROSENSCHEIN, Jeffrey S.
Computer Science Department
Stanford University
Stanford, CA 94305

SAMUEL, Arthur L.
Computer Science Department
Stanford University
Margaret Jacks Hall 436A
Stanford, CA 94305

SANNELLA, Michael
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

SAUTER, John
801128 Bates Road
Merrimack, NH 03054
603-424-7637

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

SIEGMAN, Anthony
Ginzton Lab 35
Stanford University
Stanford, CA 94305
415-497-0222

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

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

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

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

THEDFORD, Rilla J.
Mathematical Reviews
611 Church Street
P.O. Box 8604
Ann Arbor, MI 48107
313-764-7228

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

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

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

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

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

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

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

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

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

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

OFFICIAL ANNOUNCEMENTS

TUG Meeting, January 11–12, 1982, Cincinnati, Ohio

A general meeting of the TEX Users Group will be held at Stouffer's Cincinnati Towers on Monday and Tuesday, January 11–12, 1982. All members are urged to attend. For more details, see the article by Tom Pierce on page 7.

Individual Membership Dues and Privileges

1982 dues for individual members of TUG will be $15. Membership privileges will include all issues of TUGboat published during the membership (calendar) year. All new members and other persons inquiring about TUG will be sent TUGboat Vol. 1, No. 1, but after January 1, 1982, Volume 2 (1981) will be available only as a complete volume, at $10. Beginning in 1982, foreign members will be able, on payment of a supplementary fee of $12 per subscription, to have TUGboat air mailed to them.

TUGboat Schedule

The deadline for submitting items for Vol. 3, No. 1, will be February 12, 1982, a month after the Cincinnati meeting; the mailing date will be March 15. Contributions on magnetic tape or in manuscript form are encouraged; editorial addresses are given at the bottom of page 2, and a form containing instructions for submitting items on tape is bound into the back of this issue.

It is TUG's policy to keep all issues of TUGboat in print. Each member is entitled to receive all issues which appear during the membership year, as well as Vol. 1, No. 1. Domestic subscriptions are mailed third class bulk, which may take up to six weeks to reach its destination; foreign shipments are surface printed matter, unless the air mail option is elected. If you have not received an issue to which you are entitled, write to TUG at the address given on the order form for general correspondence.

* * * * * * * * * * *

General Delivery

* * * * * * * * * * *

MESSAGE FROM THE CHAIRMAN

Michael Spivak

This message is both brief and urgent. I hope that as many people as possible will be able to attend the January meeting of TUG. In the first place, there should be a great deal to report concerning progress in bringing TEX up at various installations, and in solving related problems. But even more crucial is the inescapable fact that, as Lynne Price has consistently and eloquently urged, we are clearly going to be led, almost against our wills, to adopt a more formal structure; TUG clearly cannot continue to enjoy the happy anarchistic structure of the past. To a large extent this just means that a lot of us are going to have to work harder, within committees, to draft proposals that can then be considered by the entire membership. (You say you'd like to volunteer? Why how nice!) But it also means that clearer ways of reaching decisions will have to be agreed upon. This is undoubtedly the most important (meta-)decision of all, since it will eventually affect everyone connected with TUG, and we certainly want to hear all viewpoints. If you can't come, but have any strong feelings about how TUG should function, please write to me, or any one else on the Steering Committee, so your views will be known. But if you can, please come.

* * * * * * * * * * *

Editor's note: There have been some new volunteers for Site Coordinators, as one can see from looking at the Steering Committee list. If you are installing TEX on a computer which is not yet represented, and would like to volunteer, please get in touch with any member of the Steering Committee; if your computer is already *listed, and you feel that you can offer additional support, the Site Coordinator is the person to call. It is not necessary that the Site Coordinator be the actual installer—in fact, there is considerable merit in having a Site Coordinator who is technically knowledgeable, but not actively involved in maintenance of a TeX system, since answering the questions of a large user population (potential or actual) can take quite a lot of time. To those who have already volunteered as Site Coordinators, and to those who will come forward as the TeX community grows, thanks.*

Your attention is also called to another matter of general interest: the compilation of a reference file of output devices compatible with TeX, giving their physical characteristics, where they may be obtained, and other features of interest to sites installing TeX. A questionnaire on this subject has already been mailed to all members; for more details, refer to the questionnaire and to the article by Rilla Thedford on page 14.

* * * * * * * * * * *

TUG TREASURER'S REPORT

October 31, 1981

Beginning balance, January 1, 1981:			$(419)
Income: 1981 Membership[1]		$ 3,180	
1982 Membership[2]		645	
1982 Foreign postage[2]		72	
Tape leasing		1,400	
Tape sales		800	
Workshop[3]		7,695	13,792
Current expenses[4]:			
TUGboat Vol. 2, No. 1: 500 copies			
Printing/mailing		$ 1,392	
TUGboat Vol. 2, No. 2: 800 copies			
Printing	$1,234		
Postage	575		
Clerical	66	1,875	
Reprinting TUGboat:			
Vol. 1, No. 1: 300 copies		195	
Vol. 2, No. 1: 300 copies		655	
Microfiche TUGboat: Vol. 1, No. 1; Vol. 2, No. 1		109	
Miscellaneous postage, express charges		489	
Steering Committee luncheon meeting, San Francisco, January '81		170	
Workshop expenses[3]		346	
Support for Stanford TeX Coordinator[5]		3,600	(8,831)
Estimate of future 1981 expenses:			
TUGboat Vol. 2, No. 3: 800 copies			
Printing	$1,100		
Postage	500		
Clerical	70	$ 1,670	
Questionnaire/membership renewal			
Printing	$ 100		
Mailing	150	250	
Reserve for 1981 expenses for Cincinnati meeting, January 1982		1,000	(2,920)
Anticipated receipts during the rest of 1981 against 1982 individual memberships[2]			4,000
Balance (estimate to December 31, 1981)			$ 5,622

Notes:

1. 1981 memberships number 565, of which 22 are complimentary; of the total, 406 members are domestic and 159 foreign.
2. 1982 memberships to date number 65, of which 22 are complimentary. Six members have subscribed to the $12 foreign air mail postage option.
3. The TeX Implementors' Workshop held at Stanford, May 14–15, 1981, was attended by 92 participants.
4. Not included in these figures are costs for services provided by AMS professional staff, including programming, reviewing and editing, answering telephone inquiries, maintaining the mailing list, and other administrative/clerical services.
5. Professor Arthur Samuel is acting for Luis Trabb-Pardo as TeX coordinator, answering questions, distributing tapes, and fixing bugs in the TeX source code. Luis has asked, and the finance committee has agreed, that TUG contribute to Professor Samuel's support during 1981.

Respectfully submitted,
Samuel B. Whidden, Treasurer

* * * * * * * * * * *

Editor's note: Progress continues at Stanford toward the "definitive" Pascal TeX, which we now know will be known as TeX82. The documentation system developed for the first implementation of TeX-in-Pascal, DOC, *is about to be replaced by something newer and better. We've received permission from Don Knuth to publish his "internal" status report, which we do for information only. The software described here will undoubtedly be announced formally at the Cincinnati meeting.*

STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-2085

DONALD E. KNUTH
Fletcher Jones Professor
Department of Computer Science

Telephone:
(415) 497-4367

October 13, 1981

To: Coordinators of TEX implementations

From: Donald Knuth

Subject: Current state of things

For some reason I woke up this morning with the feeling that some of you were wondering what I have been up to recently. So I decided to write a short note about present and future plans for TEX.

I'm going to be releasing the "definitive" TEX early next year. The manual will be rewritten, and all the code will also be rewritten, although of course the existing manual and programs will be pretty close to the finished product. A hardcover book will be published containing both the manual and the complete program documentation. The manual by itself will also be published separately in paperback.

Naturally I want to make sure that no bugs are present in either the manual or the system, so I will be distributing numerous preprints of all the matrial for comments and testing, until a high level of confidence has been reached.

The present implementations of TEX in SAIL and in PASCAL seem to be quite stable. With thousands of users, many of whom are quite sophisticated, no bugs have been reported for several months. The new implementation will make the present ones obsolete, of course, but we are still distributing copies of the current implementations to new users.

At the moment I am completing the development of a new documentation system call WEB. This is a greatly improved version of the 1979 DOC/UNDOC/TEXDOC system used for the first implementation of TEX in PASCAL; the new names are WEB/TANGLE/WEAVE. Basically, the WEB language describes a program's structure, using a mixture of TEX and PASCAL that can be considered as an extension of both languages to do much more than either language can do separately. The TANGLE program inputs a WEB description and rearranges everything so that a syntactically correct PASCAL program is obtained as output. The WEAVE program combines the TEX and PASCAL portions of WEB description with TEX formatting macros, so that the resulting TEX file will generate a structured documentation.

The "definitive" TEX, which will be officially known as TEX82, will of course be defined as a web. At present I'm finished with the "hardest" 20 per cent of this revision. There will be no "SYSDEP" module as in the present implementation; the system dependent parts will, however, be much simpler than they are now and their TEX-specific features will be moved to system-independent parts of the implementation.

The best way to speed up the process of TEX82 installation seems to be to prepare for it in advance, by installing the WEB system. Therefore I plan to distribute preliminary copies of the following in mid-November:

(1) WEB User Manual (hardcopy)

(2) WEAVE.WEB (the WEAVE program, in WEB language)

(3) TANGLE.WEB (the TANGLE program, in WEB language)

(4) TANGLE.PAS (the PASCAL program that TANGLE outputs when you apply it to itself)

(5) WEBHDR.TEX (macros used to print the output of WEAVE)

You need (4) for bootstrapping, after which you can change TANGLE and WEAVE as needed for your system. A few system-dependent subroutines are present in TANGLE and WEAVE, but the necessary changes are trivial compared to those in the existing TEX, so I doubt if it will take long to install this system.

(The bootstrapping process is a little interesting: First you change the system-dependent parts of TANGLE.PAS and TANGLE.WEB, until you can get TANGLE to reproduce itself. Then you make the same changes to the system-dependent parts of WEAVE.WEB. Then you can use TANGLE to create a working program WEAVE.PAS. Then you can use WEAVE to create the files WEAVE.TEX and TANGLE.TEX, from which the TEX compiler that you now have will generate beautiful documentation listing. Next year when you get TEX.WEB, your job will be to make similar changes to its system-dependent parts, after which WEAVE and TANGLE will produce the TEX.TEX and TEX.PAS files you need to get TEX82 going.)

Our preliminary experiments with software generation using WEB have proved to be quite successful, so you may in fact get some use out of this before TEX82 arrives. Of course, the WEB system will only be a few weeks old in mid-November, so you will also be able to help us diagnose any bugs that it may contain, if you are interested.

I hope to have TEX.WEB ready for testing by the end of this year and to have a draft of the new manual done by the end of January. After TEX is finished, the same will be done for METAFONT, but that will take some time.

Now I have a question for you: Do you want to be one of the guinea pigs who receive the first WEB system in November? If so, please send a letter as soon as possible to Dr. Arthur Samuel, Dept. of Computer Science, Stanford University, Stanford CA 94305, telling (a) where to send the material, (b) if you can get it over the ARPANET or if a tape should be sent. And please enclose $25.00 if you can, since that will defray our expenses of making and sending the tape.

DEK/pw

TEX USERS GROUP 1982 WINTER MEETING

Monday and Tuesday, January 11–12, 1982
Stouffer's Cincinnati Towers, Cincinnati, Ohio

A TEX Users Group meeting will be held to discuss TEX issues of general interest. The Steering Committee will also meet with the membership to discuss dues and the future development of the Users Group.

The meeting will cover three areas of interest:

- TEX-in-Pascal, with demonstrations of TEX on the Canon Laser Beam Printer,
- macro packages (both development and exchange methods), and
- output devices and interfaces.

Manufacturers of phototypesetters are invited to discuss their equipment and its TEX compatibilities. Seminars for the different computer architecture groups will be mediated by the Site Coordinators.

Donald Knuth will speak on a subject yet to be decided. Other members of the Stanford group will also be present, both as speakers and to participate in computer and output device sessions.

The first copies of "version 0" of *The Joy of TEX* (the AMS-TEX manual), are expected to be available for sale.

Pre-register early as the hotel reservation deadline is December 15, 1981. A meeting registration form is included with this issue. For additional information, contact

Thomas Pierce
'82 Winter Meeting Coordinator
Rohm and Haas Research Laboratories
727 Norristown Road
Spring House, PA 19477

* * * * * * * * * * *

Software

* * * * * * * * * * *

PASCAL-CODED TEX ERRATA

Arthur L. Samuel
Stanford University

The rate at which errors are being reported for the "PASCAL-coded version of TEX" has slowed down to the point that we seem to be approaching that famous last bug. All known errors have been corrected in the DOC and PASCAL files available from Stanford and an August 1981 revision of the descriptive documents has been printed. There are a few instances where suggested changes (for example, as in the naming of variables) have not yet been made.

One correction, made since August, is reported below. Also reported is a suggested improvement to **procedure** *PrintOctal*.

37. The following procedure should work on both 36-bit and 32-bit machines and specifically on those machines where the previous PrintOctal procedure caused overflow problems. This may be used to replace procedure PrintOctal; in section 37 on page 18 of the SYSDEP module.

```
procedure PrintOctal(n : integer); { Prints the
                rightmost 32 bits of an integer in octal }
  var i, k : integer;
    s : array [0..10] of asciiCode;
    msb, mbb : boolean;
  begin msb := false; mbb := false;
  if n < 0 then
    begin n := -n; msb := true
    end;
  for k := 10 downto 0 do
    begin i := n mod 8;
    if k = 0 then
      begin if msb then
        begin if 1 > 3 then i := 7 - i
                                   { it was a 36-bit word }
        else begin if (i mod 2) = 0 then i := 3
          else i := 2
          end
        end
      else i := (i mod 4)
      end
    else begin if msb = true then
        begin if mbb = true then i := 7 - i
                             { a borrow has propagated }
        else begin if i > 0 then
            begin i := 8 - i; mbb := true;
                                   { a borrow required }
            end
          end
        end;
      end;
    case i of
      0 : s[k] := zero;
      1 : s[k] := one;
      2 : s[k] := two;
      3 : s[k] := three;
      4 : s[k] := four;
      5 : s[k] := five;
      6 : s[k] := six;
      7 : s[k] := seven
      end;
    n := n div 8;
    end;
  Print('"'); k := 0;
  while (k < 10) and (s[k] = zero) do
    Increment(k);
  for k := k to 10 do Print(s[k])
  end;
```

453. An error has been found and fixed in the main operating module of the PASCAL-coded version of

TEX as described in section 453 on page 157 of the August 1981 revision. This error affected the use of *leqno*. The correction involves replacing the word *linq* with *link* and adding a missing line of *shift* := 0.0; as shown below:

```
453. ⟨Attach equation number 453⟩ =
  begin q := getnode(gluenodesize);
  typ(q) := gluenode; gluelink(q) := fillglue;
  if leqno then
    begin link(q) := b; link(eqnobox) := q;
    b := hpack(eqnobox, dw − shift, false);
        {eqno will be left-justified }
    shift := 0.0;
    end
  else begin link(q) := eqnobox; link(b) := q;
    b := hpack(b, dw − shift, false)
        {eqno will be right-justified }
    end
  end
This code is used in section 444.
```

* * * * * * * * * * *

THE FORMAT OF PXL FILES

David Fuchs

A PXL file is a raster description of a single font at a particular resolution. These files are used by driver programs for dot matrix devices; TEX itself knows nothing about PXL files. Let's say a user creates a file called FOO.MF, which is the **METAFONT** language description of a new font, called FOO. In order for everyone to be able to run TEX jobs that use this font and get their output on our 200-dot-per-inch proof device, we must first run the **METAFONT** program on FOO.MF, and ask it to make both a TFM file and a PXL file for it. These files (called FOO.TFM and FOO.PXL) are then put in a public directory so that anyone using TEX may access them. Now, whenever a TEX job is run that refers to FOO (`\font A=FOO`), the TEX program reads in FOO.TFM to get all the width, height, depth, kerning, ligature, and other information it needs about any font. To get output on the proof device, the DVI file produced by TEX must now be processed by a device-driver program. This program reads the postamble of the DVI file to find out the names of all the fonts referred to in the job, and for each font, it opens the corresponding PXL file. In our example, the driver would find the file FOO.PXL, which it would then use along with the main body of the DVI file to produce the actual output. The DVI file tells where to put all the characters on each page, while the PXL file(s) tell which pixels to turn 'on' in order to make each character.

In fact, there is a little lie in the preceding paragraph. The actual name of the PXL file would be something like FOO8.1000PXL. This means that the PXL file represents the font FOO in an 8-point face for a 200-dot-per-inch device with a magnification of 1. (If you don't fully understand the term 'magnification' as it is used in the TEX world, the rest of this paragraph might not make a lot of sense. The end of this document contains more information on magnified fonts.) If we also had a 100-dot-per-inch device, we would also want to have the file FOO8.0500PXL (which we can get by asking **METAFONT** nicely). This PXL file could also be used by the higher resolution device's driver for any TEX job that asked for `\font B=FOO8 at 4pt`; or one that used `\font C=FOO8`, but then got TEXed with magnification 500; or one that used `\font C=FOO8`, but then got spooled with magnification 500. Note that we are assuming that the font FOO is like the CM family in that it does not scale proportionately in different point sizes—we are only talking about the 8-point face. If it turns out that 8-point FOO magnified by 1.5 is exactly the same as 12-point FOO, then we can also use FOO12.1000PXL in place of FOO8.1500PXL, and so forth. For fonts that scale proportionately like this, a point-size should not be included as part of the font name, and FOO.1000PXL is by convention the 10-point size of FOO for a 200-dot-per-inch machine.

Now for an explanation of where the bits go. A PXL file is considered to be a series of 32-bit words (on 36-bit machines, the four low-order bits of each word are always zero). In the discussion below, "left half word" means the highest-order 16 bits in a word, and "right half word" means the 16 next-highest-order bits in the word (which are exactly the lowest-order 16 bits on 32-bit machines).

Both the first and last word of a PXL file contain the PXL ID, which is currently equal to 1001 (decimal). The second-to-last word is a pointer to the first word of the Font Directory. (All pointers are relative to the first word in the file, which is word zero.)

The general layout of a PXL file looks like this:

PXL ID	[1 word long – First word of the PXL file]
RASTER INFO	[many words long – begins at second word]
FONT DIRECTORY	[512 words – 517th-to-last through 6th-to-last word]
CHECKSUM	[1 word – fifth-to-last word]
MAGNIFICATION	[1 word – fourth-to-last word]
DESIGNSIZE	[1 word – third-to-last word]
DIRECTORY POINTER	[1 word – second-to-last word]
PXL ID	[1 word – Last word of the PXL file]

The Font Directory is 512 words long, and contains the Directory Information for all the 128 possible characters. The first four words of the Font Directory have Directory Information about the character with ascii value of zero, the next four words are for the character with ascii value of one, etc. Any character not present in the font will have all four of its Directory Information words set to zero, so the Directory Information for the character with ascii value X will always be in words $4*X$ through $4*X+3$ of the Font Directory. For example, if the second-to-last word in a PXL file contained the value 12000, then words 12324 through 12327 of the PXL file contain information about the ascii character "Q", since ascii "Q" = '121 octal = 81 decimal, and $4*81 = 324$. The meanings of a character's four Directory words are described below.

The first word of a character's Directory Information has the character's Pixel Width in the left half-word, and its Pixel Height in the right half-word. These numbers have no connection with the 'height' and 'width' that TEX thinks the character has (from the TFM file); rather, they are the size of the smallest bounding-box that fits around the black pixels that form the character's raster representation, i.e. the number of pixels wide and high that the character is. For example, here is a letter "Q" from some PXL file:

```
00 ....*******....
01 ...*********...
02 ..****...****..
03 .***.......***.
04 ****.......****
05 ***.........***
06 ***.........***
07 ***:.*****..***
08 **********.****
09 .*****..******.
10 ..****...****..
11 ...*********...
12 ..X.*******..**
13 ........***.***
14 .........******
15 ..........*****
   000000000011111
   012345678901234
```

(The rows and columns are numbered, and the reference point of the character is marked with an 'X', but only the stars and dots are actually part of the character—stars represent black pixels, and dots represent white pixels.)

Note that the Pixel Width is just large enough to encompass the leftmost and rightmost black pixels in the character. Likewise, the Pixel Height is just large enough to encompass the topmost and bottommost black pixels. So, this 'Q's Pixel Width is 15 and its Pixel Height is 16, so word 12324 in the example PXL file contains $15*2^{16}+16$.

The second word of a character's Directory Information contains the offset of the character's reference point from its upper-left-hand corner of the bounding box; the X-Offset in the left half-word, Y-Offset in the right half-word. These numbers may be negative, and two's complement representation is used. Remember that the positive x direction means 'rightward' and positive y is 'downward' on the page. The offsets are in units of pixels. In our 'Q' example, the X-Offset is 2 and the Y-Offset is 12, so word 12325 of the example PXL file contains $2*2^{16}+12$.

The third word of a character's Directory Information contains the number of the word in this PXL file where the Raster Description for this character begins. This number is relative to the beginning of the PXL file, the first word of which is numbered zero. The layout of Raster Descriptions is explained below. The Raster Descriptions of consecutive characters need not be in order within a PXL file—for instance the Raster Description for character 'Q' might be followed by the Raster Description of the character 'A'. Of course, a single character's Raster Description is always contained in consecutive words.

If a character is 'totally white' then the third word contains a zero. (The Pixel Width, Pixel Height, X-Offset and Y-Offset of a 'totally white' character must be zero too, although the TFM Width may be non-zero. A non-zero TFM Width would mean that the character is a fixed width space. TEX's standard CM fonts do not contain any such characters.) For the "Q" example, word 12326 might contain any number from 0 thru $12000 - 16$ (since Q's Raster Description is 16 words long), let's say it is 600.

The fourth word contains the TFM Width of the character. That is the width that TEX thinks the character is (exactly as in the TFM file). The width is expressed in FIXes, which are $1/(2^{20})$th of the design size. The TFM Width does not take into account the magnification at which the PXL file was prepared. Thus, if "Q" had a width of 7 points in a 12-point font, word 327 in the PXL file would contain $trunc((7/12)*2^{20})$. See the TFM documentation for more information on FIXes.

After the 512 words of Directory Information come 3 words of font information:

First, the checksum, which should match the checksum in any DVI file that refers to this font (otherwise TEX prepared the DVI file under the wrong assumptions—it got the checksum from a TFM file that doesn't match this PXL file). However, if this word is zero, no validity check will be made. In general, this number will appear to contain 32 bits of nonsense.

Next is an integer representing 1000 times the magnification factor at which this font was produced. If the magnification factor is XXXX, the extension to the name of this PXL file should be XXXXPXL.

Next comes the design size (just as in the TFM file), in units of FIXes (2^{-20} unmagnified points; remember that there are 72.27 points in an inch). The design size should also be indicated by the last characters of the PXL's file name. The design size is not affected by the magnification. For instance, if the example font is CMR5 at 1.5 times regular size, then the PXL file would be called CMR5.1500PXL, word 12513 would contain 1500, and word 12514 would contain $5 * 2^{20}$.

The word after the design size should be the pointer to the Directory Information, and the word after that should be the final PXL ID word. Thus, if the number of words in the PXL file is p (i.e. word numbers zero through $p-1$) then the Directory Information Pointer should equal $p - 512 - 5$.

All of the PXL file from word 1 up to the Font Directory contains Raster-Descriptions. The Raster Description of a character is contained in consecutive words. Bits containing a '1' correspond to 'black' pixels. The leftmost pixel of the top row of a character's pixel representation corresponds to the most significant bit in the first word of its Raster Description. The next most significant bit in the first word corresponds to the next-to-leftmost pixel in its top row, and so on. If the character's Pixel Width is greater than 32, the 33rd bit in the top row corresponds to the most significant bit in the second word of its Raster Description. Each new raster row begins in a new word, so the final word for each row probably will not be "full" (unless the Pixel Width of the character is evenly divisible by 32). The most significant bits are the ones that are valid, and the unused low order bits will be zero. From this information, it can be seen that a character with Pixel Width W and Pixel Height H requires exactly $(ceiling(W/32))*H$ words for its Raster Description.

In our "Q" example, words 600 through 615 would have the binary values shown here (high order bit on the left):

```
600 00001111111000000000000000000000
601 00011111111100000000000000000000
602 00111100011110000000000000000000
603 01110000000111000000000000000000
604 11110000000111100000000000000000
605 11100000000111000000000000000000
606 11100000000111000000000000000000
607 11100111110011100000000000000000
608 11111111110111100000000000000000
609 01111100111111000000000000000000
610 00111100011110000000000000000000
611 00011111111100000000000000000000
612 00001111111001110000000000000000
613 00000000111011100000000000000000
614 00000000011111100000000000000000
615 00000000001111100000000000000000
```

As an example of the case where the Pixel Width is greater than 32, consider a character with Pixel Width = 40 and Pixel Height = 30. The first word of its Raster Description contains the the leftmost 32 pixels of the top row in the character. The next word of the Raster Description contains the remaining 8 pixels of the first row of the character in its most significant 8 bits, with all remaining bits zero. The third word contains the left 32 pixels of the second row of the character, etc. So, each row takes 2 words, and there are 30 rows, so the Raster Description of this character requires 60 words.

Finally, some implementation notes and advice for DVI-to-device program writers: First, please note that PXL files supersede our older raster description (VNT) files. One notable difference is that VNT files claimed to have two representations for each character, one being the 90 degree rotation of the other. While a rotated copy of every character is useful in many circumstances, there is no reason that installations using only one character orientation should be burdened with so much wasted space. **METAFONT** outputs PXL files as described above, and there is a separate utility that can read a PXL file and write a rotated version into a new PXL file. Naming conventions to keep various rotations of the same font straight are currently under consideration.

Another item still under consideration is alternate packing schemes. You may have noticed that the current way that rasters are stored is fairly wasteful of space: Why should a new raster row begin in the next word rather than the next byte of the raster description? The answer is that this is for the sake of the poor people who are doing page-painting for their Versatec/Varian on their 32-bit mainframe computer. All the extra zeros help them write a faster paint program. An alternate, as yet unimplemented, PXL format would pack the rasters tighter for the sake of those who have a minicomputer dedicated to doing the painting process. Such byte-packed PXL files will be identified by a PXL ID of 1002. A straightforward utility program can convert between word-packed and byte-packed PXL files.

For those of you still in a fog about character widths, here's more prose: The intent is that a DVI-to-device program should look like DVITYP, always keeping track of the current-position-on-the-page in RSU-coordinates. DVITYP looks at TFM files in order to get the character width info necessary to interpret a DVI file in this way. The DVI-to-device program shouldn't have to open a lot of TFM files in addition to the PXL files it will be needing, so the system has redundant width information for each character—a PXL file has all of the character widths exactly as they appear in the TFM file (in units of FIXes). Thus, the DVI-to-device program can completely interpret a DVI file by getting character widths from PXL files rather than TFM files. The purpose of the CHECKSUM is to ensure that the TFM files used by TEX when writing the DVI file are compatible with the PXL files the DVI-to-device program sees (so if someone changes a font and makes new TFMs but not new PXL files, you'll have some way of knowing other than seeing ragged right margins).

In the places where DVITYP would print a message indicating that "Character C in Font F should be placed at location $\langle H, V\rangle$ on the page" (where H and V are in RSUs), the DVI-to-device program should cause character C to be put on the paper at the point closest to $\langle H, V\rangle$ that the resolution of the device allows. The important point is that the DVI-to-device program should *not* attempt to keep track of the current-position-on-the-page in device-units, since this will lead to big roundoff problems that will show up as ragged right margins.

Note again that the character widths are different things than pixel-widths: The width of the character "A" in CMR10 is (say) 6.7123 points, independent of the representation of that character on the page. For a 100-dot-per-inch device, the raster representation of "A" might be 18 pixels wide, while for a 200-dot-per-inch device, the best representation might be 33 pixels wide.

Here is some more information to help clear up misunderstandings concerning magnification. (Much of this is taken from TEX's errata list, and is destined to be included in the next TEX manual.) One point to keep in mind is that the character widths in a PXL file are exactly as in the TFM file. Since the magnification factor is not taken into account, a DVI-to-device program should multiply the widths by the product

$$\text{(font design size)} \times \text{(overall job magnification)} \\ \times \text{(font magnification)} \times \text{(254000 RSU/inch)} \\ \times (1/2^{20}\ \text{point/FIX}) \times \text{(1/72.27 inch/point)}$$

to get the effective character width in RSUs. (Of course, this multiplication should only be done once per character per job!)

It is sometimes valuable to be able to control the magnification factor at which documents are printed. For example, when preparing document masters that will be scaled down by some factor at a later step in the printing process, it is helpful to be able to specify that they be printed blown up by the reciprocal factor. There are several new features in TEX to allow for greater ease in the production of such magnified intermediate output.

TEX should be thought of as producing as output a "design document": a specification of what the final result of the printing process should look like. In the best of worlds, this "design document" would be constructed as a print file in a general and device-independent format. Printing a magnified copy of this document for later reduction should be viewed as the task of the printer and its controlling software, and not something that TEX should worry about. But real world constraints may force us to deviate from this model somewhat.

First, consider the plight of a TEX user who plans to print a document magnified by a factor of two on a printer that only handles 8.5″ by 11″ paper. In order to determine an appropriate `\hsize` and `\vsize`, this user will have to divide the paper dimensions by the planned magnification factor. Since computers are so good at dividing, TEX offers this user the option of setting the "magnification" parameter to 2000, warning TEX of the anticipated factor of 2 blow up, and then specifying `\hsize` and `\vsize` in units of "`truein`" instead of "`in`". When inputting a "`true`" distance, TEX divides by the scale factor that "magnification" implies, so as to cancel the effect of the anticipated scaling. Normal units refer to distances in the "design document", while "`true`" units refer to distances in the magnified printer output.

Secondly, some existing print file format and printer combinations have no current provision for magnified printing. This is not generally the case for DVI files, but a Press file, for example, uses absolute distances internally in all positioning commands, and Press printers treat these distances as concrete instructions without any provision for scaling. There is a program that takes a Press file and a scale factor as input and produces as output a new Press file in which all distances have been appropriately scaled. But it is inconvenient to be forced to use this scaling program on a regular basis. Instead, the Press output module of TEX chooses to scale up all distances by the "magnification" factor when writing the output Press file. Thus, the Press files that TEX writes are not representations of TEX's abstract "design document", but rather representations of the result of magnifying it by the factor (\parval12)/1000. On the other hand, the DVI files written by other versions of TEX contain normal units of distances, and the software that translates DVI files to instructions that drive various output devices will do the magnification by themselves, perhaps even using a magnification that was not specified in the TEX source program; if the user has not specified "true" dimensions, his or her DVI output file will represent the design document regardless of magnification.

Caveat: Due to the manner in which the current implementation of TEX writes Press files, it is not permissible to change the value of parameter 12 in the middle of a TEX run. If you want to produce magnified output, you should reset parameter 12 once very early in your document by using the \chpar12 control sequence, and from then on leave it alone. Another caveat below discusses the situation in more detail.

The magnification mechanism has been extended to include font specifications as well: in order to print a document that is photographically magnified, it is essential to use magnified fonts. A font is specified by the "\font" control sequence, which now has the syntax

\font ⟨fontcode⟩=⟨filename⟩ at ⟨dimen⟩.

The "at" clause is optional. If present, the dimension specified is taken as the desired size of the font, with the assumption that the font should be photographically expanded or shrunk as necessary to scale it to that size times the magnification factor specified by parameter 12. For example, the two fonts requested by the control sequences

```
\font a=CMR10 at 5pt
```

and

```
\font b=CMR5 at 5pt
```

will look somewhat different. Font a will be CMR10 photographically reduced by a factor of two, while font b will be CMR5 at its normal size (so it should be easier to read, assuming that it has been designed well).

The dimension in a font specification can use any units, either standard or "true". The interpretation of "true" here is identical to its interpretation in the specification of any other distance: asking for a font "at 5pt" requests that the font be 5 points in size in TEX's "design document", while asking for a font "at 5truept" requests that the font be 5 points in size after the scaling implied by the "magnification" factor.

If the "at ⟨dimen⟩" clause is omitted, TEX defaults the requested size to the design size of the font, interpreted as a design (non-"true") distance. Thus, the control sequence "\font a=CMR10" is equivalent to the sequence "\font a=CMR10 at 10pt", assuming that the designer of CMR10 has indeed told TEX that CMR10 is a 10-point font.

Caveat: This extension allows the TEX user to request any magnification of any font. In general, only certain standard magnifications of fonts will be available at most raster printers, while most high-resolution devices have scalable fonts. The user of TEX at any particular site must be careful to request only those fonts that the printer can handle.

Caveat: As mentioned above, you shouldn't change the value of parameter 12 in the middle of a run. TEX uses the value of parameter 12 in the following three ways:

(i) Whenever the scanner sees a "true" distance, it divides by the current magnification.

(ii) At the end of every page, TEX's output module may scale all distances by the current magnification while converting this page to format for an output device (this doesn't happen with DVI output).

(iii) At the very end of the TEX run, the output module uses the current magnification to scale the requested sizes of all fonts. Given this state of affairs, it is best not to change parameter 12 once any "true" distance has been scanned and once any page has been output.

Some device-drivers give the user the option of overriding the magnification at which the TEX job was run. Note that running a given TEX job with \magnify{2000} is not the same as running it with \magnify{1000} and then asking the driver to override the magnification to 2000. The difference will be in the dimensions of the pages; in the first case, the output will be, say, 8.5″ by 11″ pages filled with double size fonts, while in the second case the output will be 17″ by 22″ pages with double size fonts.

This is a new document, so it is bound to contain outright errors along with the portions that are merely misleading. I would certainly be glad to hear of any errors, but I am also interested in which parts of the explanation need clearing up, either by adding to or changing the text.

so mine is driven through a 300 bit per second terminal line and my APPLE II. I have constructed a protocol to compress the page image for transmission the way a FAX machine would (during a previous weekend) but even so it takes about half an hour to print a moderately complex page.

I made one change to SYSDEP and LVSPOOL for the sake of convenience. I changed all references to [TEX] to be instead TEX$:. I did this because I didn't want the administrative hassle of setting up [TEX] as a top-level directory, so instead I put TEX into a sub-directory and defined the logical name TEX$ to point to it. I recommend this to all users of TEX on VMS, since it makes using TEX more convenient.

I had one problem with TEX: I compiled SYSDEP with /CHECK, and sometimes when loading a font I get a subrange error in MAKEPTS.

I am looking forward to the "final" version of TEX for the VAX-11, which will let me run the fancy macro packages published in TUGboat, and let me build font tables for my low-resolution printer. Keep up the good work, TEX fans, and, from now on, count me in!

John Sauter

Sample of IDS-460 output – see next page

* * * * * * * * * * *

Output Devices

* * * * * * * * * * *

Editor's note: Not every TEX user has a fast, high-resolution output device at his elbow; it is surprising how modest is some of the equipment that has been made to produce usable, if limited, TEX output. The preceding page is reproduced from output on an IDS-460, an impact printer with a resolution of 84 dots per inch, driven by an Apple II. Its most severe present limitation is a lack of fonts, certainly not the ingenuity of its owner, John Sauter, who describes a recent weekend's work on page 34.

* * * * * * * * * * *

DIRECTORY OF OUTPUT DEVICES

Rilla J. Thedford
Mathematical Reviews

To further the development of TEX, TUG is compiling a directory of TEX output devices. We hope this will be a useful tool.

The directory will identify manufacturers for TEX output devices and required software and hardware for TEX interfaces. It will also include information about fonts, device specifications, contact people, and installation sites.

The membership renewal form will have additional questions for *ACTIVE* TEX users. Please complete and return it to the American Mathematical Society, with your renewal fee. Please feel free to include additional information you feel will help in the purchase and/or installation of a particular output device.

* * * * * * * * * * *

A VARIAN OUTPUT DRIVER IN VAX/VMS FORTRAN

Jim Mooney
September 8, 1981

A summer project at the Morgantown Energy Technology Center (METC), Morgantown, WV, has resulted in experimental installation of the Oregon Software version of VAX/VMS TEX. As part of this project I have written a DVI file converter to drive a Benson-Varian 200 dot/inch matrix plotter. Since METC lacks a Pascal compiler, this program was written in FORTRAN 77.

We acquired Oregon Software's TEX in July through the efforts of Tom Pierce who was then at METC. As described by Barry Smith, this version is an interim one which still uses (a kind of) TFX font files and produces version 0 DVI files. Also supplied was a pair of conversion programs LVSPOOL and LHSPOOL for a Versatec 200 dot/inch plotter. Fortunately this was very similar to the Varian. A set of character files with extension VRT were included containing the raster character descriptions for the Versatec.

The executable module supplied for TEX ran immediately and we were shortly producing DVI files. However, it was disturbing that the program seems to take between one and five minutes to initialize. (We have a VAX 11/780 with typically 30-35 interactive users.) I sometimes despaired of ever seeing that starting asterisk. I suspect that much of this time is spent in routinely loading the "standard" fonts, and since many of these fonts are often not used I hope the decision to always load them can be repealed.

An inconvenience we encountered here was caused by the OS-TEX SYSDEP module which referred to the directory "`[TEX]`" explicitly. This prevented using any other name; far worse, it made it impossible to run TEX while the default directory was on a different disk than the [TEX] directory. This could easily be fixed by switching to a logical name such as `TEX$DIR`, which we would have done if we had a Pascal compiler.

There was also a tendency for the program to abort with "`Fatal errors`" when it should have known better, e.g., when the installation limit for distinct versions of TEXOUT.DVI was reached.

The Pascal driver program LVSPOOL was supplied in a form which can be linked to a separate assembly language module to issue the actual driver calls. Thus we could have substituted appropriate QIO's for the Varian driver and obtained output. However, there were several advantages to rewriting the driver in FORTRAN. It could be locally maintained, and adapted for other output devices. Moreover, some inefficiencies found in LVSPOOL could be eliminated. For these reasons we wrote a new conversion program, based closely on LVSPOOL, in FORTRAN 77.

The Varian conversion program is called DVITOVAR. It currently translates only version 0 DVI files, but conversion to the newly announced version 1 would be straightforward. Currently at METC, the Varian interface and the [TEX] system are on two different VAX mainframes. For this reason, DVITOVAR actually creates a rather large file (1100 blocks per output page) containing raw raster data, which is transmitted to the other VAX over DECNET and converted to QIO's by the separate program OUTTOVAR. At an installation with everything on the same machine, this headache can be eliminated by inserting the QIO's directly in DVITOVAR in place of OPEN and WRITE statements. (The peculiar structure of the Varian-supplied driver program does not allow raster plot files to be spooled.)

LVSPOOL set aside almost a full megabyte to hold character raster data, far more than needed. FORTRAN does not allow the preferred solution of dynamic allocation, but we reduced the buffer to 200K bytes which is probably still lots too much. DVITOVAR also defers font loading until a font is actually needed; thus many fonts are never loaded although they are defined in the macros and thus appear in the postamble. This is a considerable timesaver, and reduces even further the buffer size needed.

DVITOVAR is rather verbose in announcing the processing phases it is going through. These messages can be removed if desired. The program has not been adapted to an equivalent of LHSPOOL which produces output horizontally on the page, but such a project should present no difficulties.

DVITOVAR was also adapted into a similar program DVITOLP to drive lineprinter class devices (Yes, many users do need such primitive output). To get this to work I had to construct with trepidation, understanding little of the format, a new TFX file to represent line printer fonts. (Font CMTT which *simulates* such a font was not satisfactory.) All widths in this font are set to 7.2 points (ten pitch); there is no kerning or ligatures; wordspace is set to 7.2 points with zero shrink, and several parameters I didn't understand were left alone. But this font seems to serve the purpose as long as all spacing parameters in the text are appropriately restricted.

Anyone interested in obtaining the programs cited above should contact

Jerry Craig
Morgantown Energy Technology Center
B1-330
Collins Ferry Road
Morgantown, WV 26505
304-599-7178

Technical questions can be addressed to me at

Dept. of Statistics and
Computer Science
West Virginia University
Morgantown, WV 26506
304-293-3607

Meanwhile, I await word of a TEX version which may be adapted to run on our PDP-11/34, which has UNIX v6 and the rather strict ISO standard Pascal from Vrije University, Amsterdam.

* * * * * * * * * * *

DIABOLIC TEX

Timothy Murphy
Trinity College Dublin

Preamble

Before TEX can be run with a given output device, 2 modules must be provided: an input module, consisting of a set of font tables; and an output module, or driver, which will translate the ".DVI" file produced by the main TEX program into instructions for the output device.

Even for a Diablo, writing these modules can prove a time-consuming occupation, at least for amateurs of the computing art like ourselves. Since our only output device was a Diablo—Versatecs and Varians being as remote from us as Neptune and Pluto—we wrote to all those in the TUG membership list under the Diablo heading. The response was disheartening; the few replies we received being from groups in much the same position as ourselves, viz Waiting for Godot.

This brief account of our own efforts may therefore not be out of place. At the very least it may shame some of the TEXperts who have well-developed Diablo drivers to share their secrets with us beginners.

The Diablo as printer

One can envisage 3 very different ways in which the Diablo might be used as an output device.

(1) The output could be run through the Diablo 2 or more times, with different daisy-wheels installed on each iteration, e.g. first with roman, then italic, then symbol, etc. The driver would of course have to be designed so that only those characters in the appropriate font were printed on each run.

(2) The output might be sent through the Diablo just once, with a single daisy-wheel, those characters not appearing on this wheel being "made up" by superposition of existing characters (moved up, to the right, etc, so as to give the required facsimile).

(3) All characters and symbols might be made up out of dots, using the graphics mode on the Diablo. In effect this would make the Diablo analogous to a digitalised type-setter, albeit one of very low resolution.

Our calculations seemed to show that the third solution would be impracticably time-consuming, each page taking more than half-an-hour to put out. We hope to implement the first solution shortly.

This could presumably give output of quite good quality. But we began by writing a driver to the second specification; and that is what is described here.

Minimal font requirements for TEX

TEX can be run with only 1 font (presumably roman) provided the text does not include mathematical formulae. If full mathematical mode is required (so that all the control sequences in the TEX manual can be used) then 10 fonts must be supplied, namely: 3 roman fonts for ordinary size, script size and script-script size; 3 italic and 3 symbol fonts similarly; and 1 font for outsize characters (including those built up from smaller parts). However, the fonts for different sizes need not really be different, e.g. roman script and roman script-script may well be the same. (But roman and italic cannot coincide, since entirely different characters occur in corresponding places on the 2 fonts.) Thus the minimal number of fonts needed is 4: roman, italic, symbol and "ex" for extra large characters.

We provide these 4, plus a "typewriter" font which allows us to print files, e.g. of macros, exactly as they are written.

Our solution: an overview ...

As remarked above, TEX requires information about the particular output device in use both on input (the widths, heights, etc of the characters) and on output (how to interpret the DVI bytes).

We keep all the information required in a single file, DIABLO.TBL. This helps to ensure that the input and output modules match: any changes made in one being accompanied by appropriate modifications to the other. Two programs, MKTFM.PAS and MKFNT.PAS, then construct the input and output information from this in the required format.

More precisely, MKTFM.PAS constructs from DIABLO.TBL the 4 font tables needed for mathematical work, DIARM.TFM, DIAIT.TFM, DIASY.TFM and DIAEX.TFM, together with the "typewriter font" DIATT.TFM. These are written in the format (FILE OF integer) required by TEX.

Meanwhile, at the "front end", MKFNT.PAS constructs from DIABLO.TBL a file DIABLO.FNT, which our output driver DVIDIA.PAS takes as auxiliary input in addition to the DVI file produced by TEX.

... and some details

The account above is somewhat simplified. In practice we have found it useful to split both the input and the output modules, so that we have a "readable" account of what is going on at each stage.

Thus for the input module we first produce a single large file containing all 4 fonts in hexadecimal form. A second program then converts this ".TFH" file into the 5 requisite ".TFM" files.

Similarly, at output we first unpack the ".DVI" file into bytes, before translating these into Diablo instructions.

We also found it convenient to split off the "constant" part of DIABLO.TBL (containing the prefaces and epilogues to the .TFM files) into an auxiliary file DIABLO.AUX. This leaves DIABLO.TBL to concentrate on the actual construction of the 640 characters in the 51 fonts.

To summarise: all font information is kept in the 2 files DIABLO.TBL and DIABLO.AUX. The program MKTFH.PAS constructs a readable file DIABLO.TFH from these; and MKTFM.PAS then converts this into the 5 .TFM files corresponding to the 5 fonts.

With these font files in place we can run TEXPRE. We are then ready to put our manuscript file, say MS.TEX, through TEX.

The program DVIBYT.PAS unpacks the file MS.DVI produced by TEX into its constituent bytes, in the readable file MS.BYT.

Meanwhile MKFNT.PAS has constructed from DIABLO.TBL a file DIABLO.FNT for our output driver BYTDIA.PAS. This driver converts the file MS.BYT into a file MS.DIA ready—at last—to be sent to the Diablo.

Command files

It would be tedious to go through the above rigmarole every time we had a file to TEX. So we make free use of command files to cut the slog.

We find the DEC-20 (TOPS-20) .MIC (Macro Interpreted Commands) file format particularly convenient, since it allows us to pass parameters—the name of the file to be TEXed, and the directory in which the TEXing is to be done.

With .MIC's help, we need only type in 2 commands. On first setting up TEX we type

```
@do texpre <scratch>
```

This installs TEX in our "public" directory <scratch>. To TEX a file, say MATHS.TEX (supposing both ourselves and this file resident in the directory <scratch>), we give the command

```
@do tex maths
```

The output for the Diablo is written in the file MATHS.DIA.

These 2 .MIC files are listed in Appendix A, since they provide a good summary of the relations between our numerous programs.

It is not necessary to study MICology in order to understand these files. Suffice to say that lines starting with @ represent commands normally entered at the terminal; while lines starting with * correspond to entries made in response to requests from within programs.

The Diablo table

Most of our time and effort has gone into 2 modules, the Diablo table and the driver.

Looking first at the table, DIABLO.TBL takes the form of a textfile, with 1 line for each of the $5 \times 128 = 640$ characters in our 5 fonts. The first 2 lines should make the pattern clear:

```
0000B   w=9    "\h3\b|\v3\u-\d \r"    \Gamma
0001B   w=10   "\h2/\v3\d---\u\\\r"   \Delta
```

The figure following "w=" is the width of the character. At present we take all characters to have the same height 6 vu, and the same depth 0 vu. (For the meaning of "vu", see the next section.) It will be easy enough to allow varying heights, etc, later, if that proves necessary. The string in quotes following the width contains the instructions for printing the character on the Diablo. The backslash introduces control sequences with the following meanings:

`\hn`	set HMI to n (i.e. $n/120$ inch)
`\r`	reset HMI to standard setting ($n = 10$)
`\vn`	set VMI to n (i.e. $n/48$ inch)
`\u`	move up
`\d`	move down
`\f`	move forward
`\b`	move back
`\",\\`	print " or \

Some of the more interesting characters in DIABLO.TBL are listed in Appendix B.

Diabolic points

The horizontal resolution of the Diablo is 1/120 inch, and the vertical resolution 1/48 inch. All movements are through multiples of these. We therefore found it convenient to introduce a horizontal unit "hu", equal to 1/120 inch, and a vertical unit "vu", equal to 1/48 inch.

For the moment we have actually re-defined "point" to have these 2 meanings, according as they refer to horizontal or vertical measure. This ensures that actual movements all take integer values, simplifying the arithmetic of width tables, etc. However, the machinery to implement proper points is all in place.

The Diablo driver

Given the format of .DVI files, the driver for a particular device almost writes itself; and indeed most of our driver is actually device-independent.

A very abbreviated version of the driver may be found in Appendix C. All PROCEDURE headings are given; but where there are several similar PROCEDUREs, only 1 body is listed. Also horizontal and vertical movements are treated in much the same way; so only one of these is detailed.

Our PASCAL compiler PASC20 allows the inclusion of header files containing CONST and TYPE declarations. This useful feature greatly reduces the risk of incompatible modifications being made to different modules. Our header file TEXDIA.H is listed in Appendix C after the driver BYTDIA.PAS.

Our only real design decision was to accumulate movements. TEX puts out a large number of redundant movements, e.g. successive DVI instructions might order an upward movement of 2 points, followed by a downward movement of 10 points. To prevent the Diablo from doing a St Vitus dance, we accumulate all movements until printing is imminent. Thus a record is kept of the point (realH, realV) on the page where the "cursor" actually is, as well as the point (H, V) where it should be, if all movements to date had been implemented.

The actual position is only updated—by making the appropriate horizontal and vertical movements—when a print instruction is received.

Appendix A. The 2 command files

```
TEXPRE.MIC

@define s: <scratch>
@copy sysdep.pas, texpre.pas, tex.pas s:
@copy ascii.tbl s:
@copy sysdep.str, texpre.str, tex.str s:
@copy texdia.h, mktfh.pas, mktfm.pas s:
@copy mkfnt.pas, dvibyt.pas, bytdia.pas s:
@copy diablo.aux, diablo.tbl s:
@copy diablo.tex, basic.tex s:
@copy tex.mic s:
@connect s:
@pasc20
*sysdep=sysdep
*texpre=texpre
*tex=tex
*mktfh=mktfh
*mktfm=mktfm
*mkfnt=mkfnt
*dvibyt=dvibyt
*bytdia=bytdia
*↑Z
@load texpre, sysdep
@save
@load tex, sysdep
@save
@delete sysdep.pas, texpre.pas, tex.pas
@delete sysdep.rel, texpre.rel, tex.rel
@delete strini.tbl
@append sysdep.str, texpre.str strini.tbl
@exe mktfh
*diablo.aux
*diablo.tbl
*diablo.tfh
@exe mktfm
```

```
*diablo.tfh
*diarm.tfm
*diait.tfm
*diasy.tfm
*diaex.tfm
*diatt.tfm
@exe mkfnt
*diablo.tbl
*diablo.fnt
@run texpre
*\input diablo \end
@delete strini.tbl
@append sysdep.str, tex.str strini.tbl
```

```
TEX.MIC

@connect <scratch>
@run tex
*\input 'A \end
@run dvibyt
*'A.dvi
*'A.byt
@run bytdia
*'A.byt
*'A.dia
```

Appendix B. Excerpts from DIABLO.TBL

```
0000B   w=9    "\h3\b|\v3\u-\d \r"        \Gamma
0001B   w=10   "\h2/\v3\d---\u\\\r"       \Delta
0002B   w=10   "\h3\b(\b--)\h2 \r"        \Theta
0003B   w=12   "\h4\b/ \\ \r"             \Lambda
0004B   w=12   "\h0/\r\\"                 \Xi
0005B   w=12   "\h3|\v8\u_\d| \r"         \Pi
0006B   w=10   "\h0>\v3\u-\d\d-\u\r "     \Sigma
0007B   w=10   "Y"                        \Upsilon
0010B   w=10   "\h0o][\r "                \Phi
0011B   w=10   "U\b|"                     \Psi
0012B   w=10   "\h00\v2\u\r_\d"           \Omega
0060B   w=10   "O"                        0
0101B   w=10   "A"                        A
0132B   w=10   "Z"                        Z
0137B   w=12   "\h6-\r-"                  --
0141B   w=10   "a"                        a
0172B   w=10   "z"                        z
0200B   w=9    "\h3\b|\v3\u-\d \r"        \Gammait
0213B   w=12   "\h4c( \r"                 \alpha
0214B   w=10   "\h3|\b0o\v3\uo\d\h7 \r"   \beta
0215B   w=0    ""                         \gamma
0216B   w=10   "\h0o\v2\u\r<\d"           \delta
0217B   w=10   "\h0<\r-"                  \epsilon
0220B   w=10   "\h0c\v3\u<\d\r "          \zeta
0221B   w=10   "\h2n\v2\d|\h6 \r"         \eta
0222B   w=10   "\h00\r-"                  \theta
0223B   w=10   "i"                        \iota
0224B   w=10   "k"                        \kappa
0225B   w=12   "\v5\d\h1'\v1\u'\v4\u\r\\"    \lambda
0226B   w=10   "\h2\b,\ru"                \mu
0227B   w=13   "\h3(\r/"                  \nu
0230B   w=9    "\h0c\v2\uc\v1\d\h1\b\h0'\v2\d\r,\v1\u"
                                          \xi
0231B   w=10   "\h0\v1\u-\v3\d\v1\d\r\"\v3\u\""    \pi
0232B   w=12   "\h2\b\v2\d|\u\ro"         \rho
0233B   w=12   "\h2o\v1\d\r}\u"           \sigma
0234B   w=13   "\v1\d\h1}\u\h2t\d\r}\u"   \tau
0235B   w=10   "v"                        \upsilon
0236B   w=10   "\h0o\r/"                  \phi
0237B   w=10   "x"                        \chi
0245B   w=10   "\h0o\r\v1\d'\u"           \partial
0260B   w=10   "0"                        0
0301B   w=10   "A"                        A
0372B   w=10   "z"                        z
0373B   w=12   "\h0/\h2-\h0\v2\d'\v4\u\r"\v2\d"
                                          \psi
0374B   w=13   "\h3u\ru"                  \omega
0400B   w=10   "-"                        -
0401B   w=10   "\v2\u.\d"                 \cdot
0402B   w=10   "x"                        \times
0403B   w=10   "\v1\d*\u"                 \ast
0404B   w=10   "\\"                       \rslash
0405B   w=10   "\v1\uo\d"                 \circ
0406B   w=10   "\h0+\v3\r\d-\u"           \pm
0407B   w=0    "\h0+\v3\r\u-\d"           \mp
0410B   w=10   "O\b+"                     \oplus
0411B   w=10   "O\b-"                     \ominus
0412B   w=10   "X\bO"                     \otimes
0413B   w=12   "\h1\v1\u,\dO\v3\u\r\d"    \odiv
0414B   w=10   "\h0O\v1\u\r.\d"           \odot
0415B   w=14   "\h2 \h0.\v3\u.\v1\u_\v4\d\h2 \r "
                                          \div
0416B   w=10   "\h0|\r\v3\u-\d"           \interc
0417B   w=10   "\ho\r\v1\u.\d"            \bullet
0420B   w=10   "\h0|\r\v1\u_\d"           \perp
0421B   w=14   "\h2 \h0\v3\u_\v1\u_\u_\v5\d\h8 \r"
                                          \eqv
0422B   w=10   "\h0<\r\v4\d-\u"           \subset
0423B   w=10   "\h0>\r\v4\d-\u"           \supset
0424B   w=10   "\h0<\r\v4\d-\u"           {\char'034}
0425B   w=10   "\h0>\r\v4\d-\u"           {\char'035}
0426B   w=10   "\h0<\r\v4\d-\u"           \preceq
0427B   w=10   "\h0>\r\v4\d-\u"           \succeq
0430B   w=10   "\v2\d}\u"                 {\char'032}
0431B   w=10   "\h0\v1\d}\r\v2\d}\v3\u"   \approx
0432B   w=10   "<"                        {\char'020}
0433B   w=10   ">"                        {\char'021}
0434B   w=10   "\h0=\r/"                  {\char'033}
0435B   w=10   "\h0=\r\v4\u.\d"           \doteq
0436B   w=10   "<"                        \prec
0437B   w=10   ">"                        \succ
0440B   w=9    "\h3<--\r"                 {\char'137}
0441B   w=9    "\h3-->\r"                 {\char'031}
0442B   w=10   "\h0|\r↑"                  \up
0443B   w=10   "\h0|\r\v1\dv\u"           \down
0444B   w=12   "\h3<--\r>"                {\char'027}
0445B   w=16   "\h6<\r<"                  \lsls
0446B   w=16   "\h6>\r>"                  \grgr
0447B   w=10   "\h0-\r\v1\d}\u"           \simeq
0450B   w=12   "\h6<=\r"                  {\char'137}
0451B   w=12   "\h6=>\r"                  {\char'031}
0457B   w=18   "\h6|-\r>"                 \mapsto
0460B   w=10   "'"                        \prime
0461B   w=12   "\h6o\ro"                  \infty
0462B   w=10   "\h0C\r-"                  \in
0463B   w=10   "\h0C-\r/"                 \notin
0464B   w=10   "\h00\r/"                  \emptyset
0465B   w=10   "_"                        _
0470B   w=12   "\h4\\-/\r"                {\char'024}
0471B   w=14   "\h0\v3\u-\d-\d\h4-\u\r|"   {\char'025}
0472B   w=0    ""                         char'5
                                      not implemented
0473B   w=9    "\h3\v3\d'\\'\r"           \aleph
0474B   w=10   "R"                        \real
0475B   w=10   "I"                        \imag
0476B   w=10   "\h0|\r\v7\u_\d"           \top
0500B   w=0    "\h0/\r"                   \not
0501B   w=10   "A"                        \Ascr
0533B   w=16   "\h8\\/\r"                 {\char'023}
0534B   w=16   "\h8/\\\r"                 {\char'022}
0536B   w=16   "\h8/\\\r"                 {\char'004}
0534B   w=16   "\h8\\/\r"                 {\char'037}
0540B   w=9    "\h3|-\r-"                 \vdash
0541B   w=9    "\h3--\r|"                 \dashv
```

```
0542B   w=10   "|"                             \lfloor
0543B   w=10   "|"                             \rfloor
0544B   w=10   "|"                             \lceil
0545B   w=10   "|"                             \rceil
0546B   w=10   "{"                             \{
0547B   w=10   "}"                             \}
0550B   w=10   "<"                             \langle
0551B   w=10   ">"                             \rangle
0552B   w=10   "|"                             \relv
0553B   w=6    "\h3||"                         \leftvv
0554B   w=6    "\h2[| \r"                      \dleft
0555B   w=6    "\h2|] \r"                      \dright
0560B   w=12   "\v4\d\h6'/\r"                  \surd
0561B   w=10   "#"                             \#
0562B   w=9    "\h0\\\h1\b\v3\u\h2----\d/\r"  \nabla
0563B   w=9    "\h3\v2\u(\d\d)\u\r"            \smallint
0564B   w=12   "\h6\b|\v1\u_\d|\r"             \lub
0565B   w=12   "\h6\b|\v7\u_\d|"               \glb
0571B   w=10   "\h0|\r\v1\u-\d"                \dag
0572B   w=10   "\h0|\v1\u-\v2\d\r-\v1\u"     \ddag
0574B   w=10   "@"                             \@
0575B   w=0    ""                              \copyright
0576B   w=12   "\h2-\rL"                       \sterling
0577B   w=10   "$"                             \$
```

Appendix C. The Diablo driver (much abbreviated)

```
PROGRAM bytdia (input, output);

INCLUDE 'TEXDIA.H'

VAR
    fnt_file: fnt_store;
    stack: ARRAY [stack_range] OF pts;
    font_mem: ARRAY [mem_range] OF byte;
    char_width: ARRAY
                  [font_range,0..127] OF byte;
    char_base:  ARRAY
             [font_range,0..127] OF mem_range;
    b, c: byte;
    H, V, x, y, z, w: pts;
    true_H, true_V, p, q: pts;
    page_no, SP, i: integer;
    BS, HT, LF, VT, FF, ESC, RS, US: char;
    f: font_range;
    printing, overprinting: boolean;

FUNCTION hu_from_pts (p: pts): integer;
BEGIN  hu_from_pts := round (p)  END;

FUNCTION vu_from_pts (p: pts): integer;

FUNCTION hu_to_pts (hh: integer): pts;
BEGIN  hu_to_pts := hh  END;

FUNCTION vu_to_pts (vv: integer): pts;

PROCEDURE hmi_set (b: byte);
BEGIN  write (ESC, US, chr (b+1))  END;

PROCEDURE vmi_set (b: byte);

PROCEDURE hmi_reset;
BEGIN  hmi_set (hor_spacing)  END;

PROCEDURE vmi_reset;

PROCEDURE hor_tab (b: byte);
BEGIN  write (ESC, HT, chr (b+1))  END;

PROCEDURE vert_tab (b: byte);

PROCEDURE initialise;
BEGIN
    BS := chr(8);  HT := chr(9);
    LF := chr(10);  VT := chr(11);
    FF := chr(13);  ESC := chr(27);
    RS := chr(30);  US := chr(31);
    page_no := 0;  SP := 0;
    hmi_reset;  vmi_reset
END;

PROCEDURE read_2_bytes (VAR p: pts);

PROCEDURE read_3_bytes (VAR p: pts);

PROCEDURE read_4_bytes (VAR p: pts);

VAR  c, d, e, f:  byte;

BEGIN
        read (c, d, e, f);
        p := c*256 + d + (e + f/256)/256;
        IF (c >= 128)
        THEN p := p - 256*256
END;

PROCEDURE move_to (H, V: pts);

VAR  xx, hh, hhq, hhr, yy, vv, vvq, vvr:
                                     integer;

BEGIN
    xx := hu_from_pts (H - true_H);
    IF (xx <> 0)
    THEN    IF (abs(xx) < 127)
        THEN    BEGIN
                hmi_set (abs(xx));
                IF (xx > 0)
                THEN write (' ')
                ELSE write (BS);
                true_H :=
                    true_H + hu_to_pts (xx);
                hmi_reset
            END
        ELSE    BEGIN
                hh := hu_from_pts (H);
                hhq := hh DIV 64;
                hhr := hh MOD 64;
                hmi_set (64);
                hor_tab (hhq);
                hmi_set (hhr);
                write (' ');
                hmi_reset;
                true_H := hu_to_pts (hh)
            END;
    yy := vu_from_pts (V - true_V);
    IF (yy <> 0)
  ...
END;
```

```
PROCEDURE hor_line_length (p: pts);

PROCEDURE vert_line_length (p: pts);

VAR  yy, yyq, yyr, i:  integer;

BEGIN
    yy := vu_from_pts (V + p - true_V);
    yyq := yy DIV 4;  yyr := yy MOD 4;
    hmi_set (0);  vmi_set (yyr);
    write ('|', LF);  vmi_set (4);
    FOR i := 1 TO yyq DO write ('|', LF);
    V := V + p;
    true_V := true_V + vu_to_pts (yy);
    hmi_reset;  vmi_reset
END;

PROCEDURE push_stack;

PROCEDURE pop_stack;

BEGIN
    IF (SP < 6)
        THEN writeln (tty, 'Stack exhausted');
    w := stack[SP];  SP := SP - 1;
    z := stack[SP];  SP := SP - 1;
    y := stack[SP];  SP := SP - 1;
    x := stack[SP];  SP := SP - 1;
    V := stack[SP];  SP := SP - 1;
    H := stack[SP];  SP := SP - 1
END;

PROCEDURE new_page;

BEGIN
    write (FF);
    H := 0;  V := 0;
    true_H := 0;  true_V := 0;
    page_no := page_no + 1
END;

PROCEDURE store_font (VAR fnt_file: fnt_store);

VAR  i:  integer;  b:  byte;  f:  font_range;

BEGIN
    i := 0;  b := 0;  f := 1;
    WHILE NOT eof (fnt_file) DO
    BEGIN
        char_width [f,b] := fnt_file↑;
        get (fnt_file);
        char_base [f,b] := i;
        REPEAT
            font_mem [i] := fnt_file↑;
            get (fnt_file);
            i := i + 1
        UNTIL (font_mem [i-1] = 0);
        b := (b + 1) MOD 128;
        IF (b = 0) AND NOT eof (fnt_file)
        THEN f := f + 1
    END
END;

PROCEDURE change_font
            (VAR f: font_range; ch: char);
```

```
BEGIN
    IF (ch IN ['r', 'i', 's', 'e', 't'])
    THEN    CASE ch OF
                'r':    f := 1;
                  ...
                't':    f := 5
            END
    ELSE writeln (tty, 'Undefined font ',
                                    f, ' used')
END;

BEGIN   (* main *)
    initialise;
    reset (fnt_file, 'DIABLO.FNT');
    store_font (fnt_file);
    WHILE NOT eof AND (b <> 131) DO
    BEGIN
        read (b);
        IF (b <= 127)
        THEN
            BEGIN
            IF NOT printing
            THEN    BEGIN
                    move_to (H, V);
                    printing := true
                END;
            i := char_base [f, b];
            WHILE (font_mem [i] <> 0) DO
            BEGIN
                write (chr (font_mem[i]));
                i := i + 1
            END;
            IF overprinting
            THEN    BEGIN
                    printing := false;
                    overprinting := false
                END
            ELSE    H := H + char_width [f,b];
            true_H := true_H + char_width [f,b]
           END
        ELSE    IF ((128 <= b) AND (b <= 153))
            THEN
                BEGIN
                printing := false;
                CASE b OF
                    128:    ;    (* NOP *)
                    129:    BEGIN    (* BOP *)
                            FOR i := 0 TO 10
                            DO read_4_bytes (p);
                            new_page
                        END;
                    130:    ;    (* EOP *)
                    131:    ;
                        (* start of postamble *)
                    132:    push_stack;
                    133:    pop_stack;
                    134:    BEGIN
                                (* vertrule *)
                            ...
                        END;
                    135:    BEGIN
                                (* horzrule *)
                            read_4_bytes (p);
                            read_4_bytes (q);
```

```
                              hor_line_length (q);
                              H := H - q
                          END;
                    136:  BEGIN
                              overprinting := true
                          END
                    137:  BEGIN  (* font *)
                          END;
                    138:  BEGIN
                              read_4_bytes (w);
                              H := H + w
                          END;
                          ...
                    END
                END
        ELSE    IF ((154 <= b) AND (b <= 217))
                THEN change_font (f, chr(b-90))
    END
END.
```

The header file TEXDIA.H

```
CONST
    hor_spacing = 10;       (* standard HMI *)
    vert_spacing = 8;       (* standard VMI *)
    stack_size = 125;
    mem_size = 3000;
    max_font_no = 5;

TYPE
    byte = 0..255;
    half_word = 0..65535;
    oneoftwo = 1..2;
    oneoffour = 1..4;
    halves2 =   PACKED RECORD
                lhword: half_word;
                CASE oneoftwo OF
                    1:(rhword: half_word);
                    2:(byte2: byte; byte3: byte)
            END;
    bytes4 =    PACKED RECORD
                byte0: byte;
                byte1: byte;
                CASE oneoftwo OF
                    1:(rhword: half_word);
                    2:(byte2: byte; byte3: byte)
            END;
    memoryword =    PACKED RECORD
                CASE oneoffour OF
                    1:(pts: real);
                    2:(int: integer);
                    3:(twohalves: halves2);
                    4:(fourbytes: bytes4)
            END;
    pts = real;
    stack_range = 0..stack_size;
    mem_range = 0..mem_size;
    font_range = 1..max_font_no;
    fnt_store = PACKED FILE OF byte;
    font_type = (rm, it, sy, ex, tt);
    fontfile = FILE OF memoryword;
```

* * * * * * * * * * *

Site Reports

* * * * * * * * * * *

NEWS FROM THE HOME FRONT

David Fuchs
Stanford University

Here's what's going on TEX-wise at the CS Department at Stanford. Professor Knuth has a working version of the UNDOC macro processor written in its own language (DOC). UNDOC compiles itself into a Pascal program, thus UNDOC is now available in Pascal. DOC is being used as the source language for new versions of TEXDOC and TEX82. All three programs (both DOC and Pascal sources) are expected to be available for porting to new machines in early 1982. TEX82 is a complete rewrite of TEX based on the experience gained from Ignacio Zabala's translation of Sail-TEX. Portability has been improved by removing all floating point operations. Another sticky portability problem with the current Pascal TEX is initialization. Recall that installing a new TEX involves running the program TEXPRE, which makes a large file (called TEXINI.TBL) that represents the initial state of TEX's data structures (about 36K words in size). On TOPS20, we then run TEX, which reads in TEXINI.TBL, at which point we interrupt the process and save the current core image. When our users ask for "TEX", they get a copy of this core image, which continues execution from where we interrupted the first TEX run. Thus, our users are saved the not-insignificant overhead of data structure initialization. The resulting core image is also smaller and faster than if the initialization functions of TEXPRE were to be incorporated into TEX. Unfortunately, we have found that the facility of "saving an interrupted job's core image for later continuation" is not available in many environments, including VAX VMS, UNIX, and IBM timesharing systems. Consequently, TEX users outside of the DEC 36-bit world have TEX re-read TEXINI.TBL each time it is run, which is a significant time handicap. To help rectify the situation, TEX82's data structures will change to require less initialization. We also plan to make a program available that can read TEXINI.TBL and produce Pascal-language initialization code to be inserted into the TEX Pascal source before compiling. Unfortunately, variable initialization is not standard Pascal, so there must be different versions of this program for the Hedrick compiler, Pascal/VS, VMS Pascal, etc.

Of course, the option of reading in a significantly smaller TEXINI.TBL each time TEX is run will still be a possibility, and will only use standard features of Pascal.

Scott Kim's INVERSIONS has appeared in bookstores. The text was typeset with TEX, using fonts leased from the Alphatype Corporation.

A few people in our department spent some of their summers porting Pascal TEX to a number of new machines. Joe Weening has a version working on the Cray, with output to a Versatec. Jeff Rosenschein brought up TEX in Israel on an IBM processor running VM/CMS, with output on a Versatec. I spent some time writing Pascal drivers for the Autologic APS-5 and Mergenthaler Linotron 202. In fact, it's a single program that compiles under either IBM's Pascal/VS or Hedrick's Pascal for DECSystem 10/20, and will convert DVI files to APS-5 DCRTU Input Command Language format, or Linotron CORA-V format, or Linotron Binary Byte format, depending on the setting of some compile time switches. The first version of this program is up and running at two IBM and one DEC20 sites. Each of these installations is running its typesetter with the native fonts (auxiliary programs serve to convert font width information as supplied by Autologic and Mergenthaler into TEX-compatible TFM format). This restricts the number of TEX's features that can be used, however, since information about 'height' and 'depth' of characters is not provided by either company, and can only be guessed at heuristically, and changed manually if the need arises. Also, much of math mode is crippled, since the TEX math fonts are not available. There is still some hope for the future, though. There has been some interest expressed by individuals both at Autologic in LA and Linotype-Paul in England in Knuth's Computer Modern fonts. Similarly, at the instigation of the folks in Wisconsin, Compugraphic seems interested in the possibility of providing the CM fonts to 8600 users.

One other thing I worked on over the summer was PXL files. A PXL file is the raster description of a font at a given size and resolution (such as CMR10 at 200 dots per inch). The documentation for PXL format is contained elsewhere in this issue (p. 8). Our spooling software for Varians/Versatecs has been updated to use PXLs. PXL files supersede the older VNT format and have the following advantages:

(1) They include sufficient information to be used with TEX's magnification features.

(2) I have added a PXL mode to **METAFONT**, so that PXL files can be made directly for **METAFONT** fonts.

(3) The internal layout of PXL files has been improved to allow for more efficient operation of spoolers that must deal with them.

(4) The layout is also such that PXL files can be written sequentially from **METAFONT**, which will aid in writing a transportable version of **METAFONT**.

* * * * * * * * * * *

TEX UNDER THE NORTH STAR

Michael J. Frisch
University of Minnesota

I have made some progress on the CDC Cyber TEX at the University of Minnesota since the last TUGboat, but at this writing (October 1) it is not ready for distribution. There are a few known bugs in my version of TEX and a possibility that more bugs will show up during further testing. (I believe that most of the bugs are in my implementation rather than in the original Stanford version.) As a result, I can't give an estimated release date.

I have a working—though probably not debugged —device driver program for our Varian plotter. The device driver is written in machine-dependent FORTRAN since I can write and debug code much faster in FORTRAN. The driver translates the DVI file into line segments. Each segment has the same baseline and the same size font. At the end of each page, the driver sorts the segments by vertical position and then outputs scan lines to the plotter one at a time. This allows multiple column text input to be processed.

The driver is small enough to run in interactive mode though my version of TEX will not. Luis Trabb-Pardo pointed out that if I can get TEX to run in a large memory area, then I could make a large driver keep an entire page of plotter bits in memory and vastly simplify the driver. For lack of time, I haven't tried this idea. The driver I wrote works and could be improved or rewritten later. My thanks to David Fuchs for his Pascal DVI file printing program which explained a lot of mysteries to me about the file.

My version of TEX now accepts almost all of the standard BASIC file. This is a great improvement over the TEX version I mentioned in the last TUGboat. My TEX doesn't read all the font information files correctly, because some were not properly converted from PDP-10 36-bit floating point to CDC 60-bit. Despite this, and with some temporary changes, I have made a very short TEX input file run and have produced a plot so I have hopes that I will make more progress.

Though my version of TEX is not totally functional, the University of Aarhus in Denmark has one they built that works quite well. Erik Bertelson and I recently made an oral agreement that Minnesota will distribute their version in the U.S. With code modifications and decreased memory size, their version runs in about 38K words. They have printed a couple of books using it so it is quite well debugged. They have a Pascal version of UNDOC which has made it a lot easier to change TEX. When I receive a copy of their TEX, I will install it and then decide on distribution details.

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

A TUGboat TOUR: EXCERPTS FROM THE TEXNICIAN'S LOG

Barbara Beeton
American Mathematical Society

The morning mail is very exciting around the TUGboat deadline. In it appear manuscripts and mag tapes from all over, bearing who knows what news, but certainly containing something unexpected and interesting. Putting together an issue is also fun; the readers get to see only the printed pages, but I get to grub around in the mud that seeps into all the cracks.

During this tour I'll be referring to various articles that have appeared in TUGboat, so you might want to get out your back issues. In particular, I'll be making a few changes to the previous TUGboat guidebook: *How to Prepare a File for Publication in TUGboat*, vol. 2, no. 1, pp. 53–54.

Although this *How to* ... guide states that `basic.tex` and all the features of $\mathcal{AMS}$-TEX are available, as well as some formatting macros especially for TUGboat, some authors build their own macro set from scratch, including formatting details. This may be required by local font and device limitations (see the article by John Sauter, p. 34, and the excerpt from his output, p. 13), or the author may simply have his own idea (different from mine) of how his article should look in print.

Whenever possible, I try to accommodate the author's ideas, so long as they do no violence to the articles which will follow when the issue is put together. Often this can be accomplished very simply: quarantine an article from the rest of the issue by putting { braces } at the beginning and end. (This was necessary for Brendan McKay's article in vol. 2, no. 2, pp. 46–49; any attempt to revise the format would have taken too long, and would certainly have resulted in unfortunate errors.)

Another technique is to lay out the pages in such a way that the "singular" item begins on a new page, and deal with it separately, adjusting page numbers as required. This is obviously necessary for material received as camera copy, but another reason would be that, between the standard TUGboat set and the author's, there are too many control sequences to fit in the hashtable. (Mike Plass' article on syntax chart macros, p. 39, is an example of this condition.)

While I'm on the subject of the hashtable, I should note that, here at the Math Society, we've found it necessary to change several of the size values that show up in the error message

```
! TEX capacity exceeded
```

(TEX manual, p. 144). The most important are the following: `hashsize=1009`, `memsize=32768`, `varsize=11500`. (One or more of these changes may by now have been made in the "official" SAIL version at Stanford.)

In the SAIL implementation of TEX, `memsize` is restricted to a maximum of 2^{15}; this will be lifted in Pascal TEX, but we won't be using that until the "definitive" version is published. `varsize` is effectively a subset of `memsize`, and therefore the actual capacity of `memsize` is really only about 21,200. (All these values represent 36-bit words on our DECSystem 2060.) The original value of `varsize` was 17,000, but much of our work involves large pages of small type and, being unable to increase the absolute value of `memsize` to avoid exceeding it, we had to increase its capacity by reducing `varsize`.

`hashsize` can be no larger than 2^{10}, and Don Knuth recommended that its assigned value be a prime (it used to be 797); he suggested 1,009, equal to roman `MIX`, of which he is particularly fond since he finished TEXing Volume 2 of *The Art of Computer Programming* (TEX manual, Appendix E, pp. 161 *ff.*). These changes must be made in the SAIL (or Pascal) source, and the modules recompiled.

To provide more facts on just how much of `hashize`, `varsize` and `memsize` are required by troublesome jobs, we've also created a "diagnostic" version of TEX, which reports at the end of each completed (output) page and at each occurrence of \ddt the current and maximum (so far) demands on `varsize` and `memsize`. It also, if requested, prints each control sequence name as it is loaded into the hashtable, counting down as it goes from the initial value of `hashsize` (1,009 less however many control sequences were preloaded). Some of this reporting facility already existed in the SAIL source code obtained from Stanford, requiring only that a switch, `MSTAT`, be turned on before compilation; we've suggested that a similar facility be included in the "definitive" Pascal TEX.

As TEX comes into being on more different kinds of computers, these computers generate tapes containing input files to be processed somewhere else. TUGboat gets a good sampling. We've successfully read tapes from the following computers: VAX (running under both VMS and UNIX), IBM 370, PDP-11 (TEX was not running on this machine; it was used only to prepare the tape), Univac 1100, and DECSystems 10 and 20. The DECSystem tape utilities, of course, generate tapes that are "native" to our 2060. Finding a common format for exchanging tapes with other machines has been more of a challenge.

The tape format we have settled on works quite well, although it is limited to the standard ASCII character set, one character per tape character (8 bits), which does not accommodate font files in internal form. This format is simply a variation on the old "card-image" format, with 80-character records, 100 records per block (8,000 characters per block). When reading one of these tapes, we assume that no carriage return/line feeds are present, and that all terminal spaces can be stripped from each record. We therefore recommend that input lines not end with the \␣ control sequence, since it cannot be distinguished from \⟨cr⟩ after stripping; in practice, this has occurred only rarely, and was easy to correct, although inconvenient.

Another tape problem is what to do about labels. The utility program we use to read "foreign" tapes isn't very clear about label formats, so the easiest thing to do is omit labels. We have just successfully read a IBM 370-generated tape with "ANSI standard labels" (which are not among those defined in the utility manual, and for which Susan Plass had to try long and hard to find a description, and make several attempts before a good tape was actually written); we finally treated the labels as a separate file, which was discarded once the real file was safely on disc. Suggestion: forget about labels. (They're probably really necessary only for multi-volume tapes, and no single TEX file should be that big anyhow—it would probably be big enough to contain the entire *Encyclopædia Britannica*.)

Finally, there is the matter of identifying what is on the tape. It would be appreciated if every tape were accompanied by a transmittal form (one is bound into every issue of TUGboat), and by a list of what files the tape contains. And, just to be sure,

`%` ⟨file name⟩

as the first line of each file will make it easier to check the disc copy.

Once files are safely on disc, properly identified, they undergo some editing (as little as possible) to ensure that they conform to TUGboat requirements. Of particular importance for compatibility is the use of a "standard" font set. For TUGboat (in fact, for all the Society's TEX work), our standard set is based on the one used for the "book format" (TEX manual, Appendix E, p. 152) except for `\font ?=cmti10` as in `basic.tex`. Only font codes `G` through `Z` are free to be used for job-specific fonts, so if a TUGboat author has special font requirements he should note them in comment lines at the beginning of his file and if possible identify them by letters `G`–`Z`.

To permit automatic cross-referencing and insertion of page numbers in the Table of Contents, a reference is added just following the title which allows page numbers to be sent to a separate file. An error in the second T-of-C page in vol. 2, no. 2, should give you an idea how this works (when it's done right), but here are the details anyhow. In each file, a line is inserted:

```
\pagexref{filnam}
```

which invokes the definition:

```
\def\pagexref#1{\send9{~def~#1{\curpage}}}
```

At the beginning of a TUGboat run the page number file from the previous run is read in (`~` is `\chcoded` to type 0 for the duration, so that these definitions really do become control sequences, then back to type 12="other"). Then a new version of the file is opened for output, to record new page numbers should any changes have taken place. Actually, two different file names are used so that the old file is available after the run for comparison to the new version; also, some items don't ever get run through TEX, and their references are added to the page number file manually. When the page numbers converge, and a final scan of Varian copy turns up no obvious blunders, the .DVI file is shipped off to the Alphatype for camera copy.

The main TUGboat header file is used to format the one- and two-column pages. At present, the two-column routine uses the `\save5\page` ... `\box5\hfil\page` technique. I would really like to be able to write out each column as a separate page, but the output drivers require that each "sheet" contain the same number of "pages", and I haven't been able to figure out how to output two "pages" for a single-column page. Suggestions are welcome; I'll submit this as a problem for the next issue if no solution has been found by then.

The address list uses a different header file, which does output each \page (i.e. column) separately to the .DVI file, putting the running heads and footers out as part of the last column. The width of each partial page is the distance from the left boundary of column 1 to the right boundary of the current column. This width is recalculated for every \page; \xcol is the column number, \xcolmax is the total number of columns per page, \xcolwd is the number of points each column is wide, and \intercol is the number of points skipped between two adjacent columns:

```
\def \howwide{\setcount8\xcol
     \setcount3 0
     \sowide}
\def \sowide{\advcount8 by -1
     \advcount3 by \xcolwd
     \ifpos8{\advcount3 by \intercol
            \sowide}
     \else{\xdef\thiswide{\count3 pt}}}
                  . . .
\output{ . . .
     \howwide
     \if \xcolmax\xcol{⟨output headers⟩}
     \vbox to size{
          \hbox to \thiswide{\hfil\page}}
     \if \xcolmax\xcol{⟨output footers⟩
              \gdef\xcol{1}}
     \else{⟨add 1 to \xcol⟩}}
```

I even use this to calculate the total page width (needed for, e.g., running heads; TEX's arithmetic is more reliable than mine):

```
\def \xcol{\xcolmax}
\howwide
\xdef \pagewd{\thiswide pt}
```

I've used this technique to put together pages of up to 6 columns; if \xcolmax gets much larger, Mike Spivak's \result trick (vol. 2, no. 2, p. 50) has to be used to avoid a nesting level error on \sowide.

The .DVI file now contains \xcolmax pieces for each publishable page, each of which has the same reference point (upper left corner). It merely remains for them to be overlaid by the output device driver, which is assumed to have "pasteup" capability.

Yet another header file is used to prepare the Errata list. You may already have noticed that the current one has been TEXed, unlike those sent out with previous issues. There's a good reason for the delay: all the previous schemes for encoding "typewriter-style input" have been rather cumbersome, and I've been waiting for one that's really easy to use. Mike Spivak has made up one for his AMS-TEX manual (*The Joy of TEX*) that makes input strings look just like ordinary math coding; the only difference is that *...* goes around in-line statements, and

```
**
 . . .
**
```

goes around displayed material. This will eventually become available for use in the main part of TUGboat, and Mike has promised to publish his macro.

I said I'd be changing the instructions for using some of the TUGboat control sequences described in the old *How to ...* guide. Here are the changes.

A new control sequence, \hpar will give you an indented paragraph like this one. Unlike the other hanging indented paragraphs listed, it doesn't have to be \ended.

- The control sequence \endhpar turns out to be unnecessary, since hanging indentation is transient, disappearing at the end of the current paragraph.
- The line breaks in \textaddr can now be indicated by \\; I got tired of misspelling \lbrk and made life easier (inspired laziness must be the greatest source of creativity known to man).

Actually, there have been lots more changes to the header files as I've learned more about TEX, but most of these changes are entirely transparent to the casual TUGboat author. When I'm entirely pleased with the package, I'll submit it for publication in the Macros column. (It does depend on the AMS-TEX macros, and is distributed with them on tape.)

In a later issue I'll report on the timing statistics for TUGboat. There have been many questions regarding how much computer time is required to run TEX and prepare output on various devices. Although TUGboat is probably atypical, it's one publication that is readily available to all TEX users, and thus suitable for an example.

So keep those tapes and letters coming—this is *your* newsletter, and only your participation will keep it afloat.

* * * * * * * * * * *

TEX FOR THE HP3000

Lance Carnes

Amazingly enough, the mighty TEX-in-Pascal system runs on the modest Hewlett-Packard minicomputer.

The TEX-in-Pascal tape was received from Stanford in May 1981 and it took approximately two person-months to do the conversion. There were no problems of significant magnitude compiling the Pascal sources. As with every conversion the sources had to be edited to weed out the "non-standard" features (e.g. OTHERS:) and many of the SYSDEP modules had to be rewritten. The .TFM font files converted nicely from the FIX representation of reals.

By far the most difficult task was shoe-horning TEX into a 16-bit word, 32K address space, non-virtual memory machine. Accessing the 49152 records of MEM (takes 200K 16-bit words) and the other large arrays was accomplished through liberal use of software-implemented virtual memory. A version of the Pascal P4 compiler was modified so that when a large array is referenced, code is generated to bring in chunks of the array from disc storage.

Naturally, a heavy performance penalty is paid with this implementation. Currently it takes two to three minutes to compile a single, simple page of text. Additional optimizations will be implemented before distributing this version, sometime in the next month (November 1981). Anyone with ideas and/or experience optimizing such an implementation is welcome to write to the address below.

This author has agreed to be the site coordinator for the HP3000. If you are interested in obtaining a copy of TEX for the HP3000, please write to the address below. Indicate which model and MPE release you have, and any output device(s) at your site. The initial release is scheduled for December 1, 1981.

Lance Carnes
163 Linden Lane
Mill Valley, California 94941

* * * * * * * * * * *

TEX FOR THE IBM 370

Susan Plass
Stanford Center
for Information Technology

It's finally up and running—Pascal TEX for the IBM 370 running in the MVS batch. We have successfully produced device independent (DVI) output files which have been correctly printed on the Stanford Computer Science Department Dover printer. We do not have output drivers for printing these files from an IBM 370—we hope to write those soon.

What we do have, however, is a version of PTEX which runs on our 3033 under MVS and under ORVYL, Stanford's timesharing system. The following is a brief outline of the changes we have made to PTEX in order to compile under Pascal/VS, to run on an IBM processor under MVS, and to fix a few known CSD version bugs. We have made no changes to TEX specifically to run under ORVYL; that was achieved merely by compiling TEX with an interactive version of Pascal/VS.

1. TEX

a) All integer subsets were changed to `PACKED` integer subsets.

b) All `REAL`s were changed to `SHORTREAL`.

c) `EXTERN` was changed to `EXTERNAL` throughout.

d) `OTHERS:` was changed to `OTHERWISE` throughout.

e) `INITPROCEDURE` was changed to `PROCEDURE INIT` and a call to `INIT` was added as the first executed statement.

f) In `DEFINEFONT`, the label 0 on the statement

```
LABEL 50, 31, 0
```

is never referenced, causing a warning from Pascal/VS; this label was removed.

g) In the loop from `N+1 to 30` within `LEXICALORER`, the initialization of `TRUNCWORD[I]:=0` was added to the existing initialization of `HYPHENATIONWORD`.

2. TEXPRE

Changes (a) through (e) of TEX above are identical for the TEXPRE module.

f) There are a few lines longer than 72 characters. These were broken up rather than expand the `MARGINS` option to 80.

g) At the start of `XENT` the statement `SWORD := NULWORD` was added.

h) The following were added to `INITSUF`:

```
VAR I : INTEGER;
FOR I:=0 TO 115
          DO SUFFIX[I].ALPHASET:=[];
```

i) The following were added to `INITPREF`:

```
VAR I : INTEGER;
FOR I:=0 TO 108
          DO PREFIX[I].ALPHASET:=[];
```

3. SYSDEP

Changes (a) through (d) of TEX and TEXPRE are identical for SYSDEP; there is no `INITPROCEDURE`, so `CONSTANT VALUE` variables were changed to `STATIC` data.

e) The `TYPE CHAR9` was changed to `STRING(9)`.

f) All files were changed to type `TEXT`.

g) `ASCII` translation in and out and to and from `EBCDIC` was added via the chrX and ordX arrays.

h) Terminal I/O was modified so that files would not be `RESET` for each reference (necessary for a BATCH environment).

i) `RESET` and `REWRITE` statements were modified for PASCAL/VS.

j) The routine `INTOUT` was replaced with an algorithm more suitable for `HEX`adecimal arithmetic.

k) Octal computations and constants were changed to HEX.

l) A POSTAMBLE file called PST is created in CLOSEOUT to make this information easier to access using standard IBM access methods.

m) The DVI file is always named DVI so that the same DDname may be used in batch JCL.

n) Code relating to DIRECTORY references was commented out; it is not needed in PASCAL/VS.

o) The FONT file directory was changed to a VS partitioned data set.

A tape containing the source for our version of PTEX, TEX/370, will be available soon. For details on obtaining a copy write to:

Susan Plass
C.I.T. Systems
Polya Hall, Room 203
Stanford University
Stanford, CA 94305

When available, TEX/370 will be supplied on an IBM standard-labeled 1600 bpi tape—please don't send us a tape.

We plan to fix bugs, implement new releases, and incorporate comments and criticisms into TEX/370 and will publish those changes periodically to users who have ordered TEX/370. No promises are made or implied about responses outside of such newletters, but we do welcome feedback and will try to act on it. We also plan to implement output drivers for several output devices attached to our 3033/3081. These will be announced as they are implemented.

* * * * * * * * * * *

TEX IN ISRAEL

Jeffrey S. Rosenschein
Stanford University

Over the past summer, TEX was brought up at the Weizmann Institute of Science in Rehovot, Israel. It is running there on an IBM 4341 under CMS, with the Imperial College Pascal P4 compiler, and producing output on a Versatec 1200-A.

Many of the issues addressed in this implementation of TEX have been treated (repeatedly) in previous issues of TUGboat, with regard to other machines and other versions of Pascal; nevertheless, for the sake of completeness, I will briefly outline the major points of interest. It should be noted, however, that this was an old version of TEX, received from Stanford in January 1981.

Editor's note: David Fuchs comments, "It is unfortunate that Jeff had to use an old version of TEX-Pascal. Most of the following points had already been cleared up by Ignacio Zabala and Eagle Berns while Eagle was working on compiling TEX under Pascal/VS. Because Pascal/VS packs records and has an OTHERS construct, it seems more suitable for the current TEX than the P4 compiler. TEX82 is currently planned to require OTHERS, but even the current TEX makes no assumptions regarding packing. The routine READFONTINFO is system-dependent, and the documentation in TEX82 will be more explicit about exactly which bits go where.

(1) In Imperial College Pascal, there is no default case for CASE statements; instead, an IF-THEN-ELSE construction was used to perform its function.

(2) CASE statement selector variables being out of range caused a Pascal crash (this is not the case in Stanford's Pascal). An IF-THEN construction made sure CASE statements were accessed only when the selector variable was in range.

(3) Labels that were declared but not used had to be removed.

(4) The INITPROCEDURE construct does not exist in Imperial College Pascal; instead, a procedure called INITIALIZE was introduced in its place.

(5) Overly large procedures had to be split for compilation to succeed. In the TEXPRE module, these procedures were INITIALIZE, INITMATHCODES, INITFONTCODES, INITSUF and INITPREF. In the TEX module, the procedures JUSTIFICATION and MLISTTOHLIST had to be split.

(6) Imperial College Pascal does not allow assignment between variables of differently named (though identically defined) types. Thus, the TYPE declarations of PCKDHYPHBITS, PCKDCONSPAIR and TBLREADOUTTYPE in the TEX and TEXPRE modules were changed, respectively, to declarations of PACKEDHYPHENBITS, PACKEDCONSONANTPAIRENTRY and TABLEREADOUTTYPE so as to be compatible with the corresponding SYSDEP declarations.

(7) The name INPUTFILE was used in TEX both as a procedure name and as an identifier of an enumerated type. To allow compilation, the identifier name was changed.

(8) FILES OF ASCII had to be changed to FILES OF CHAR.

(9) The ORD and CHR functions in Imperial College Pascal map to and from the EBCDIC character encoding scheme. This conflicted with TEX's expectations of an internal ASCII

encoding of all characters. Two translation arrays were utilized to convert characters to and from ASCII, thus satisfying TEX's needs.

(10) PRINTOCTAL was altered so as to work on a 32 bit machine.

(11) The procedure CONNECT was used to link internal Pascal file names to real-world files, replacing TEX's multiple parameter RESET and REWRITE procedures.

(12) All code that looked for an "end-of-line" character (usually a carriage return) was changed to utilize the EOLN function. This was necessary due to the record-oriented structure of IBM files. Likewise, instead of writing a carriage return onto a file to signify an end-of-line, the procedure WRITELN was used to finish off a record and transfer it to a file.

(13) Imperial College does not pack records as expected in the SYSDEP module code. To overcome this, the PTEXINI.TBL file was changed from a file of INTEGER to a file of MEMORYWORD, extra routines were introduced to build correct font data structures, and bytes were explicitly packed into integers for the DVI file.

(14) The SCANNUMBER routine in TEX and TEXPRE makes no check for overflow as it reads in a number from the user's TEX input file. If the hapless user includes too large a number, Pascal crashes, and there is no way of knowing that the overflow was not internal to TEX (i.e. some previously undiscovered bug). A check was introduced in the SCANNUMBER routine so that if overflow is about to occur, the ERROR procedure is called. This gives a standard dump of the buffer and allows the user a graceful recovery.

Due mainly to Imperial College Pascal's lack of record packing facilities (which causes MEMORYWORDs to each occupy 4 words of memory), it is necessary to have a runtime storage allocation of approximately 2000K to run TEX. Production of raster files for the Versatec takes about 700K. As of this writing, the system has been put through a series of relatively small tests, and (so far) seems to be working without difficulty.

As with all those who have brought TEX up at various installations, I have several suggestions for changes to the code; these are intended solely to aid portability, and increases in portability may, of course, be purchased at a cost to some other important consideration (such as efficiency). Nevertheless, if TEX-in-Pascal is really intended to be a portable program, there ought to be more consideration of standard Pascal features, and sensitivity to differing machines. The two main suggestions are:

(1) The OTHERS construct should be removed once and for all; it has no place in code that is advertised as portable, especially not in the actual TEXPRE and TEX modules (as opposed to the SYSDEP module). Its removal was time consuming, and it should not have to be carried out repeatedly at various installations. At times, finding the correct values to use in the IF-THEN construct was non-trivial due to nesting of CASE statements and the appearance of labels within procedures. In addition, there was an abnormally high chance of an error creeping into the code; such an error would be extremely difficult to track down.

(2) Assumptions about packing should be removed from the code; experience has shown that this restructuring is quite feasible. Although this will result in slight degradation of TEX's efficiency at some installations, it will cause TEX itself to be implemented with much greater ease. This is especially crucial in the READFONTINFO routine, where insufficiently explained assumptions about packing led, initially, to a serious implementation error. To help increase efficiency at those installations with "correctly" packing compilers, helpful hints on how to convert certain code should be included with the documentation; the default, however, ought to be code that will run even in a non-packing environment.

* * * * * * * * * * *

TEX AT THE UNIVERSITY OF MICHIGAN SUMMARY OF PROGRESS

David Rodgers

The University of Michigan Computing Center has installed TEX on an IBM 370/148 running under VM(CMS). Work continues on converting the system-dependent module to run under the MTS operating system on an Amdahl machine. The installation process would have gone unhindered, except for our inexperience with Pascal, the VM/CMS operating system, and the system editor. The entire installation process required about three weeks of full-time effort (spread over six weeks) and probably could have been done in half the time by an experienced Pascal/VM/CMS programmer.

A device driver has been installed for a Linotron 202 phototypesetter in the Ann Arbor area and we are evaluating options for supporting other proof-

and final-quality output devices available in the University community or through local vendors.

We are especially interested in exploring the feasibility of using intelligent terminals (e.g., ONTEL) as a source of local intelligence for driving printing devices. If anyone has had any experience with this sort of output configuration or wants to express a prejucide, we would welcome an opportunity to discuss them. Please write

David L. Rodgers
University of Michigan
Computing Center
1075 Beal Avenue
Ann Arbor, MI 48109

* * * * * * * * * * *

VAX ON UNIX

Bob Morris
UMASS/Boston

The Computer Science Department at Cal Tech has succeeded in running TEX compiled under the Berkeley VAX UNIX Pascal Compiler. By the time you read this, I hope we will have CIT's work at test sites and be working toward a clean public distribution through one or another traditional UNIX distribution arrangement. The work at CIT was done by Calvin Jackson, whose report I have included below, slightly edited. Following Calvin's report is one on device driver work at Brown University.

* * * * * * * * * * *

TEX AT CALTECH

Calvin Jackson

We have PTEX running on the VAX/780 at the Computer Science Department of CalTech. The operating system is Berkeley UNIX Version 4.0. We have compiled with Version 4.1 and corrected a minor problem—an octal constant > maxint.

We acquired a version of PTEX via the ARPANET in May 1981 after the TEX workshop at Stanford. In July we had DVI output consistent with Stanford test cases. A new version of TEX was retrieved from Stanford in July, and approximately 6 hours was required to make the local modifications and produce acceptable test results.

The Department has a DEC-20 and a VAX/780. Output devices are a Trilog C-100 (100bpi, VAX), XGP (200bpi, DEC-20), Applicon plotter (125bpi, tape), and various HP plotters. Some specialized graphics terminals are also available.

Our experiences with PTEX are similiar to those reported in previous issues of TUGboat. We agree with many published suggestions regarding better tutorial information about SYSDEP concepts. We found that most of our problems were due to lack of experience with Pascal and machine dependencies in Pascal data representation. The DOC listings were essential; we are impressed with their quality and completeness.

Compilation Problems, Source Configuration

After reviewing the DOC listing and some code samples in the UNDOC version we performed some tests to determine how some potential problem areas are handled by the Berkeley compiler. After this analysis we made some basic decisions. A preprocessor would be developed to perform some desired source transformations, separate compilation units were desired, and the SYSDEP program would be divided into smaller compilation units. A simple "lex" program was prepared to do the source transformations; it is referred to as "pedit" in the following discussion. The decision to use a preprocessor was strongly influenced by the regularity of the UNDOC version; if some other source form is used the preprocessor might not be so simple. The separate compilation-unit decision was based on compilation time and the expectation that we would be doing a lot of compilations of areas of SYSDEP.

Following are the observations and the approach adopted to cope.

The type allocation model used by Berkeley is the following:

char: 8bits. This type will perform as expected as long as the value range used is 0..127.

real: 64bits.

integer: 8, 16, or 32bits. Integers are signed (as stated in the Report). The signed attribute means that a subrange of 0..255 requires 16 bits.

Variant records are optimal within this definition.

This model allocates 64bits for the type MEMORYWORD. Note that 64bits would be required if reals were shorter. Storage economy requires short reals and signed subrange types (or the treatment of certain integers as unsigned).

The DVI file is byte oriented and is defined as the subrange type 0..255. We changed this definition to be −128..127. The DVI output routine and other routines that interface to a DVI file convert values > 127 to value −256 on output and convert values < 0 to value +256 on input. TFM files are handled on a similar basis.

In case-statements, if none of the case-constants is equal to the case-index, a runtime error is sometimes reported. My experience is that the results are unpredictable.

PTEX uses a compiler specific feature for default (OTHERS) case-constants. We decided to use the following construct.

```
  if expression in set-constructor then
    case case-index of
    ...
    end else begin
    { ...  OTHERS code follows}
    ...
    end
```

This process is performed semi-automatically by **pedit**. A program scans the source for case-statements, inserts the skeleton if statements, and generates the set-constructor on a auxiliary listing. We then manually edit in the set-constructor, the program also provides a line number directory of all case-statements.

We did some after the fact timing studies and determined that the preferred transformation is

```
      CASEARG:=case-index;
        if not (CASEINDEX in
            [LOWERLIMIT,UPPERLIMIT])
        then CASEARG:=LOWERLIMIT-1;
        case CASEARG of
        ...
        LOWERLIMIT-1,otherslist:
        ...  {OTHERS code}
        end
```

This provides acceptable efficiency, does not make an exception of the case-index being a function call, may require a few extra editing keystrokes, and is compatible with the program that scans and alters the source.

One procedure (VARSYMBOL) contained some inaccessible code due to the omission of a case-constant. The instance was reported by the compiler. We edited in the appropriate case constant. This condition did not exist in the second set of sources we retrieved (July 1981).

Equivalence of type requires that each variable be defined by the same type declaration. This was significant because of our decision to use the separate compilation feature. We manually reviewed the programs for common type declarations and placed them on an "include" file. Our first copy of PTEX (May 1981) contained instances of the same types with different spellings. A subsequent copy (July 1981) had resolved these problems. It was interesting that we had guessed wrong on the eventual spellings. This process was not very time consuming, there are surprisingly few type definitions and the lexical order of the program simplifies the task.

The empty-statement is incorrectly handled in the following construct:

```
if boolean-expression then empty-statement
else statement
```

A diagnostic is provoked. A "begin end" or system procedure "null" in place of the empty-statement will satisfy the compiler. This task was assigned to the program "**pedit**".

Labels that occur in the label-declaration-part must occur in the statement-part. Labels that did not occur in the statement-part were edited out of the statement-declaration-part. The diagnostics provided by the compiler were the cues.

The standard compilation mode treats upper-case as different from lower-case. In the standard mode keywords are lower-case. A compile option permits this treatment to be suppressed; the option specifies that warnings should be produced for non-standard Pascal. However, we could not use the separate compilation capability if the "standard" Pascal option was selected. We assigned the task of converting keywords and standard identifiers to **pedit**.

Separate compilation units are supported by the compiler. The compile system rigidly enforces the concept that the fragmented program be equivalent to its composite model. This enforcement is primarily at the compilation and linkage stages.

We manually reviewed the compilation text and constructed the following compilation units. The choice was primarily based on the DOC version, experience has suggested better fragmentation.

texpre.p: Vanilla TEXPRE.PAS except for the deletion of common subroutines and types.

tex.p: Vanilla TEX.pas except for the deletion of common subroutines and types.

sysdep.p: SYSDEP.PAS (DOC) Section 16, 64 – max, min and output routines.

basicio.p: SYSDEP.PAS Section 23 – Basic I/O procedures.

string.p: SYSDEP.PAS Section 17 – The String Handler.

filename.p: SYSDEP.PAS Section 51 – Scanning File Names.

infont.p: SYSDEP.PAS Section 57 – Reading Font Information.

fetchdata.p: SYSDEP.PAS Section 71 – Retrieving Data Structures.

storedata.p: SYSDEP.PAS Section 69 – Storing Data Structures.

globsysdep.h: SYSDEP.PAS Global variable and procedure declarations for SYSDEP.

globconst.h: SYSDEP.PAS Global constant-definition-part.

globtype.h: SYSDEP.PAS Global type-definition-part.

globexternal: SYSDEP.PAS Global procedure and function definitions.

The program-declaration must include the file "**output**".

Program declarations were edited to conform to the Report.

Variables declared to be structured types that require more than 64,536bytes of allocated space provoke a diagnostic. Berkeley was contacted regarding the limitation on the size of structured-types. My understanding is that the limitation should only apply to the use of those variables, not to their allocation. The diagnostic was not present in Version 4.1; we were provided a fix for Version 4.0. The fix required the deletion of the diagnostic and rebuild of the compiler.

PTEX uses the compiler-specific initialization feature INITPROCEDURE not supported by the Berkeley compiler. INITPROCEDURE was changed to a proper procedure and a call placed in the statement-part of the program.

Goto-statements in procedures ERROR and QUIT provoked a diagnostic during the assembly phase. The statements were sufficiently displaced from their target that the preferred instruction could not be used (its displacement field is too short). An assembly option is available that requests the longer jump. We decided to move procedure QUIT adjacent to the program-statement-part and change the goto-statement in ERROR to an invocation of QUIT.

Support Tools

Stanford and Berkeley were extremely helpful when support was requested. We particularly appreciate the efforts of I. Zabala (Stanford) and P. Kessler (Berkeley).

We used a screen editor based on EMACS and many of the standard UNIX tools. A "lex" program was invaluable for performing simple source analysis and transformation. The Stanford programs DVITYP, PLTOTF, and TFTOPL were also invaluable for understanding the interfaces and as roots for other programs. Following is a list of auxiliary programs and their function.

DVITYP: Stanford program that provides a formatted dump of a DVI file. Converted to VAX without the random access feature. Essentially no program changes required.

DVIDEC: Stanford program that provides a raw dump of a DVI file. No program changes required.

tfmdec: Program based on TFTOPL but lacking the robustness and elegance of output. Produces a raw dump of a TFM. Operates on the DEC-20 and VAX; primarily used to prepare TFM files for transfer to the VAX.

tfmbin: Program based on PLTOTF but lacking the robustness. Produces a TFM file from the output produced by tfmdec.

tfmprt: Program based on TFTOPL but lacking the robustness and elegance of output. Produces a satisfactory dump of a TFM file.

textoucb: Converts a VNT file to the Berkeley "vfont" format.

ucbtfm: Generates a partial TFM file from a Berkeley "vfont" file.

dvitriplot: Converts a DVI file to the format required for plot mode printing on the Trilog printer. Font files are in the Berkeley "vfont" format. A page of text is set and output to the standard printer spooler (raw mode option).

dvitrilog: Converts a DVI file to the format required for standard printing on the Trilog. The font used must be a typewriter style defined for the Trilog. This program is unsatisfactory.

pedit: Lex program that scans a .PAS source and discards blank lines. Inserts code to assist editing of case-statements with OTHERS. Transforms certain empty statements to "begin end". Changes keywords and standard identifiers to lower case. Produces a directory of case-statements with a complementary set-constructor. Produces a directory of procedure and function declarations. This program makes significant assumptions about the nature of the input, i.e., it is not general purpose. If there are significant changes in the format of the distributed version, the program would be discarded or changed.

Plans

We are thoroughly dissatisfied with what we have for user interface with respect to file specification. Better user control of the name of the output DVI file is desired. This work is local to two of the SYSDEP compilation units.

The system response to error conditions is not satisfactory. I believe some of this is due to the response to runtime language violations. Some is due to the way the system is designed. Experience and experimentation will, hopefully, provide some guidance.

The Versatec printer is being interfaced to the VAX. Scan conversion will occur on the VAX. Software has to be found or developed for this device. *Some from Brown is reported elsewhere in* *this issue...ram* A similar condition exists for our Applicon plotter. The Applicon is off line, data is transferred via magnetic tape.

The device interfaces are not very efficient or robust. Improvements in these areas will be pursued.

Currently, there are a number of pieces that must be used to perform the typesetting task. They have not been integrated into a reasonable user oriented capability. This has to be corrected.

Various macro packages look interesting and should be available to our users. Acquisition and installation will be pursued. (We do not yet have any TEX users.) The system is still in a pre-release state. Release and user support/encouragement are key items.

We desire to have equivalent facilities on the DEC-20, an effort has been started to host PTEX on that machine.

We plan to retrieve **METAFONT** and attempt to host it on the DEC-20.

* * * * * * * * * * *

THE STATUS OF VAX/TEX AT BROWN

Janet Incerpi

We received a tape from Calvin Jackson containing some TFM files and the TEX source and binary. We were able to get TEX running simply by reading the tape and putting the files in the directories specified. Since we had already installed VNT files and debugged software to take DVI files and send output to the Benson Varian (see TUGboat vol. 2 no. 1 p. 49) we were able to print papers within a few days. (It was necessary to change the programs DVIVER and VERSER to handle the new DVI file format.) We have not recompiled TEX, we are just running the binary we received but don't expect recompiling to be a problem.

* * * * * * * * * * *

VAX/VMS SITE REPORT

M. C. Nichols
Sandia National Laboratories

There are now 40 sites that have received the Oregon Software VAX/VMS version of TEX. The VMS group is presently in a holding pattern while deciding whether to bring up an intermediate version of TEX (the current VMS version is the one which existed at Stanford as of 11/80; it uses tfx fonts rather than tfm ones) or to wait for the most recent Pascal version. Rumor has it that the most recent version will go a long way toward eliminating the 15–20 sec. delay now experienced at the start of every TEX user's run.

Until now, the Versatec was the only output device for TEX on the VMS system, but an accompanying article by Jim Mooney (see p. 13) discusses the birth of a Varian driver written for the VAX in FORTRAN 77 (*FORTRAN STRIKES AGAIN!*). It is hoped that the Varian driver will appear on the VAX/TEX distribution tape soon along with any other drivers etc. that have yet to be publicized. Several people have called expressing interest in spoolers for the Linotron 202 and APS-5 for the VAX.

The current version of TEX for the VAX is still available from Oregon Software for a $50 reproduction charge (see TUGboat vol. 2 no. 2, page 33). Their address is

Oregon Software
2340 SW Canyon Road
Portland, OR 97201

You are to be encouraged to share this tape with other VAX users in your area, but please contact Barbara Beeton (AMS) or myself if you obtain a VMS version in this manner so others in your area will be able to find out that you have TEX up and running.

I would like to solicit news specific to VAX/VMS for possible inclusion in a TEX/VAX/VMS memo to be sent to VAX/VMS users early next year. Especially welcome would be problems, comments, programs, or macros that bring to light or solve any of the difficulties with TEX that are peculiar to the VAX/VMS system.

* * * * * * * * * * *

ENHANCEMENTS TO VAX/VMS TEX AT CALMA

John Blair
Calma

Calma has had TEX running on its VAX for about four months now, using a Versatec V–80 as an output device. While we were happy to have TEX inhouse, there were a few inconveniences to the VAX version of TEX which prompted me to rework some of the VAX-dependent code in Barry Smith's latest release of the VAX Pascal version of TEX.

The biggest problem was the fact that DEC Pascal can only open files for exclusive use, while TEX requires shared, read-only access for the initialization and font files, as well as common source files such as `basic.tex`. Thus if two people tried to run at the same time, the second (and third ...) were informed that the initialization file TEXINI.TBL was locked by another user. The quick and dirty fix for this problem (well, it wasn't really that quick, and all-in-all isn't really that dirty, considering the alternatives) was to write a set of FORTRAN subroutines and functions: FOPEN, FCLOSE, FRESET, FGET, ... to simulate the Pascal I/O calls for all input files. Since DEC FORTRAN supports shared, read-only access, the problem was solved. The only negative factor in the conversion is that initialization now takes 30 seconds elapsed time on an otherwise idle system, rather than the previous 20 seconds. I feel that this time could be shortened dramatically if I could find a free week to re-write the entire initialization procedure (using an assembler routine to block-load the data directly into the program arrays).

Previous to doing this rewrite, we worked around the problem of not being able to run concurrent images of TEX by creating a batch job queue which ran TEX jobs one at a time. To do this we changed TEX to scan the command line and automatically \input the file name found there. For example,

```
TEX foo
```

is equivalent to running TEX and then entering \input foo as the first command. For the batch system to run properly, foo.tex should contain everything from the \input basic to the \end.

A side effect of this organization is that once the TEX job is submitted (a command file handles all of the details), the terminal can immediately be used for other work, rather than having to wait for TEX to finish.

When the background job has finished, a message is broadcast to the user's terminal, and, if TEX exited normally, the .DVI file is automatically sent to the spooler. (Note that any TEX error which prompts the user for terminal input will cause TEX to exit abnormally, since the batch SYS$INPUT will indicate an end-of-file condition to the program).

The names TEXOUT.DVI and ERRORS.TMP which were hard-coded into the source were changed to DVIFILE and ERRFILE so that the command procedure could set the name through

```
assign/user foo.dvi DVIFILE
assign/user foo.err ERRFILE
```

which gives the output files the same name (but different extensions) as the input file.

Now that multiple images of TEX can run concurrently, we no longer need the batch system, but, in fact, most people still use it instead of the interactive version due to the savings in time. We could also use subprocesses to run TEX, but the batch system allows us the flexibility of specifying how many jobs can run concurrently (three or four TEX jobs running together tend to slow the system down a bit).

We had a problem using the Versatec printer due to the fact that normal print jobs and TEX jobs would both try to write to the device at the same time. This was solved without writing a symbiont manager by running both types of jobs in a single batch job queue which only let one job at a time have access to the device.

The file name parsing routines contained the error of assuming that the first period "." encountered was the delimiter between the file name and the extension. This resulted in the inability to use subdirectories in file name specifications. Fixing this allowed us to organize the [TEX] directory a bit by moving all of the fonts to [TEX.FONTS], and using [TEX.INPUT] as the default directory to try if an input file is not found in the user's directory.

As an aid to the interactive user, I put in the hooks to the VAX help utility. If the user enters Control-H in his input line, the user is passed to the help routine. When the user exits help, the command line is automatically reinstated up to the Control-H, and the user can continue the line. The usefulness of this can be appreciated by anyone who has tried to look up the syntax for a control sequence in the TEX manual. Trying to find the right page can be frustrating. Though we have the hooks for help, we currently lack the content. If anyone has a machine-readable base of information on TEX syntax and usage, I would certainly appreciate hearing from you.

Another problem which I rectified was the fact that all fonts which were defined were written into the postamble, whether they were actually used or not. Since basic.tex defines about a dozen fonts, the spooler spent a very long time loading in fonts which were not used. This was easily corrected, and was the only modification which envolved TEX.PAS itself, all other changes being contained in SYSDEP.PAS.

Barry's programs LVSPOOL (which I renamed TEXSPOOL) and READDVI were also modified to open "DVIFILE" rather than "TEXOUT.DVI". TEXSPOOL was changed to refer to the Versatec as "LVA0:" rather than "LPA0:" since we had already assigned LPA0: to our line printer. TEXSPOOL had a minor error in that it would crash if you tried to print an undefined character from a font. In general this doesn't really matter, but it makes the production of a font book quite difficult. A trivial fix took care of the problem.

Future work involves speeding up TEX initialization (perhaps I'll just wait for the new Pascal version from Stanford, which supposedly will eliminate the need for much of the initialization), as well as using Calma's interactive graphics systems to dump raster images in the form of a font so that we can incorporate drawings and shaded images directly into the TEX output for our internal documentation needs.

I'm sending off a tape of my modifications to Barry Smith so that he can implement those that he feels are worthwhile into his future releases. Anyone interested should also feel free to contact me directly.

* * * * * * * * * * *

"POOR MAN'S" TEX

John Sauter

Greetings to all of you in TEX-land, from grand wizard to humble worker in the field. I have joined your ranks, I have become a TEX user!

I am running what must surely be characterized as a "poor-man's" TEX system. Although my host machine is a VAX-11/780, my printer is an IDS-460 with the graphics option, which provides me with 84 dots per inch. The printer was tax-deductible because I also use it to help in the preparation of W2 forms on mag tape (but that's another story). After buying the printer I couldn't afford the $50 for Pascal TEX from Barry Smith, so I convinced a department at DEC that they should order it. They did and, through the magic of DECnet, I got it from them.

Putting up TEX was a real nostalgia trip. My first computer job was with the Stanford A. I. Project in the late 1960's, and when I was there I felt that any project could be completed in 6 weeks, and anything short of a major compiler could be done in a single weekend if one were sufficiently dedicated. Since leaving Stanford I have learned that even a trivial change to a text editor can take a year to get into customer's hands.

It was with great joy, therefore, that I spent last weekend making TEX work on my IDS-460. Actually, TEX itself worked with few problems. The struggle was with the post-processor to send the .dvi file to the printer. Even though I had LVSPOOL before me as an example getting everything straightened out wasn't simple. Here are the most critical things I learned, in case somebody should face a similar problem in the future:

First, TEX seems to believe that the printer operates from high bit to low bit, left to right across the page. This is the PDP-10 convention. The PDP-11 convention is to operate from low bit to high bit. The Versatec seems to follow neither of these conventions, operating from low byte to high byte, but within a byte operating from high bit to low bit. There is some code at INSERTLV to scramble a bit string from PDP-10 format to Versatec format, and my statements above about what the Versatec expects are based on it (I have never seen a Versatec manual).

My printer follows the PDP-11 convention. I considered modifying INSERTLV to convert from PDP-10 format bit strings to PDP-11 format, but decided not to, since it would involve reversing the whole string, one bit at a time. Instead I just ORed the bit string into the scan line using the VAX-11 instructions EXTZV and INSV to operate on all 32 bits at once, and arranged to scan the string in reverse order when sending it to the printer. My code for INSERTLV ended up looking like this:

```
.ENTRY INSERTLV,^M<R2>
SUBL3 @HORIZ(AP),#2080,R2
EXTZV R2,#32,@SCAN(AP),R0
BISL @BITS(AP),R0
INSV R0,R2,#32,@SCAN(AP)
RET
```

Since I don't have Metafont I was unable to convert the fonts for an 84 by 84 resolution printer, so I must pretend that my printer has 200 by 200 resolution and a small page. After some experimenting I found that a design size of 2.5 inches wide by 3 inches high works. Using the ten-point fonts I can print posters, so I have made some copies of "The Typographical Error" and am beginning to distribute them among the writers to try to generate interest in TEX. Maybe I can find someone who is willing to invest in a real printer!

I hear someone asking, "I didn't know you could interface an IDS-460 to a VAX-11." The IDS-460 is certainly not supported by DEC as a printer on the VAX-11 so mine is driven through a 300 bit per second terminal line and my APPLE II. I have constructed a protocol to compress the page image for transmission the way a FAX machine would (during a previous weekend) but even so it takes about half an hour to print a moderately complex page.

I made one change to SYSDEP and LVSPOOL for the sake of convenience. I changed all references to [TEX] to be instead TEX$:. I did this because I didn't want the administrative hassle of setting up [TEX] as a top-level directory, so instead I put TEX into a sub-directory and defined the logical name TEX$ to point to it. I recommend this to all users of TEX on VMS, since it makes using TEX more convenient.

I had one problem with TEX: I compiled SYSDEP with /CHECK, and sometimes when loading a font I get a subrange error in MAKEPTS.

I am looking forward to the "final" version of TEX for the VAX-11, which will let me run the fancy macro packages published in TUGboat, and let me build font tables for my low-resolution printer. Keep up the good work, TEX fans, and, from now on, count me in!

John Sauter
801128 Bates Road
Merrimack, NH 03054
603-424-7637
Computer: VAX-11
Output Device: IDS-460
Application: TEX posters

Editor's note: The hard copy of this report, printed on the aforementioned IDS-460, is too appealing not to share with the readership. One page (of $4\frac{1}{2}$) is reprinted as an exhibit on page 13.

* * * * * * * * * * *

Fonts

* * * * * * * * * * *

FONT UPDATE

Ronald Whitney

Probably the most important development concerning fonts since the last issue of TUGboat is the limited access that TEX users now have to the Autologic and Mergenthaler fonts (see the article by Fuchs, page 21). While this is very good news to those with APS-5's and Linotron 202's and at least a point of leverage to others, progress is still limited with regard to establishing a library of device independent font descriptions. Work on Euler at Stanford and a Cyrillic alphabet at the AMS is continuing, and other sites (including Autologic) have expressed an interest in **METAFONT**. Of course, not only alphabets but also other special fonts need our attention (see the article by Murphy, page 37). We hope that further developments and other ongoing research concerning fonts and **METAFONT** will be reported to the TEX community via TUGboat and TUG meetings.

* * * * * * * * * * *

Warnings & Limitations

* * * * * * * * * * *

Don't Just \let TEX Hang—\raise or \lower It

Beware of repeating a \let statement; TEX may hang, leaving you no alternative but to kill the run, leaving you with no error file to help in your diagnosis.

These four lines cause TEX to hang:

```
\let\oldsize=\size
\def\size{\oldsize}
\let\oldsize=\size
\size
```

These don't:

```
\let\oldsize=\size
\def\size{\oldsize}
\size
```

Nor do these:

```
\let\oldsize=\size
\xdef\size{\oldsize}
\let\oldsize=\size
\size
```

The above results were originally reported by Mike Spivak using the SAIL version of TEX. They have been duplicated by Lynne Price using the Pascal version on a VAX. An especially insidious variation has just occurred at the Math Society: A job \input a header file wherein

```
\let\italcorr=\/
\def\/{\unskip\italcorr}
```

was lurking. A second header file, essentially a copy of the first, was also \input, and somewhere *much* later on in the data, an italic correction (\/) was applied. After questioning whether the system had crashed (not an impossible occurrence), the user came to the local wizards for help. Diagnosis took a considerable length of time, and would have taken even longer had we not been familiar with the problem from Mike's experience.

Not that anyone would try it, but \raiseing and \lowering boxes by negative amounts (in the SAIL version, at any rate), has no different effect from using the same positive amounts. The following example says exactly what it's trying to do.

baseline$^{\text{raise 3pt}}{}_{\text{baseline}}{}^{\text{raise –3pt}}{}_{\text{baseline}}$

baseline$_{\text{lower 3pt}}{}^{\text{baseline}}{}_{\text{lower –3pt}}{}^{\text{baseline}}$

Barbara Beeton

* * * * * * * * * * *

MACRO COLUMN

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

The macro column is off to a good start. Readers are commenting on published macros as well as sending in descriptions of problems they have encountered. However, no one has yet submitted solutions to posed questions ...

Errata

There are two minor corrections to macros published in TUGboat, Vol. 2, No. 2.

(1) Mike Spivak notes a subtle error in McKay's definitions for ↑ and ↓. Since the macros classify these characters as type 13, spaces after them are not ignored, even in math mode. As a result

`2↑ x`

gives

$$2x$$

because the superscript is the space. For AMS-TEX, Mike has fixed this problem by having ↑ first check whether #1 is a space, using `\compare* {#1}`.

(2) When used in horizontal mode, Patrick Milligan's `\Apply` macro may cause extra space to be inserted. The carriage return after the opening left brace, and the space after #1 in the redefinition of `\Func` are significant.

* * * * * * * * * * *

TUGBOAT MACRO INDEX

The following list catalogues macros that have appeared in earlier issues of TUGboat. Entries are listed by volume, number, and page as well as author's name. Items that could not be categorized by an obvious headword have been listed under "miscellaneous". Many items refer to parts of large macro packages; users of other packages may find them valuable models for macros of their own.

Readers' comments on the format as well as the contents of this index are welcome.

ACM style	II:1	61, 82–83	A. Keller
Addresses	II:1	54	B. Beeton
	II:2	A-35	M. Díaz
Appendices	II:2	A-21	M. Díaz
Baseline, set to top of box	II:1	60, 77	A. Keller
Bibliography	II:2	A-25	M. Díaz
Boxes	II:1	59, 73	A. Keller
Branching, see If			
Capital letters			
large ~ at beginning of paragraph	II:1	60, 78	A. Keller
	II:2	A-16	M. Díaz
Roman numerals	II:1	120–121	P. Milligan, L. Price
Centering a sequence of lines	II:2	A-13	M. Díaz
Chapters and Sections	II:1	60–61, 79–81	A. Keller
	II:1	111-118	L. Price
	II:2	A-8–9, 20-22	M. Díaz
Characters, macros to produce special	II:1	57, 67–70	A. Keller
Columns, multiple	II:2	A-38–40	M. Díaz
Comparison of integral values	II:1	119–120	P. Milligan, L. Price
Counters, pseudo	II:1	60, 77	A. Keller
	II:1	120	P. Milligan, L. Price
Deferred output	II:1	60, 86–86	A. Keller
Division	II:2	47	B. McKay
Equality of integral values	II:1	119–120	P. Milligan, L. Price
Figures	II:2	A-25–27	M. Díaz
Font			
declaring families of a particular point size	II:1	56–57, 65–66	A. Keller
	II:2	A-11	M. Díaz
definition	II:1	119	P. Milligan, L. Price
	II:2	44–45	P. Milligan
Footnotes	II:1	58, 71–72	A. Keller
	II:2	A-24–25	M. Díaz
French	II:2	A-12	M. Díaz
Graphics	II:2	48–49	B. McKay
Headings, page	II:2	A-23–24	M. Díaz
If			
comparison of integral values	II:1	119–120	P. Milligan, L. Price
groupless \if	II:2	46	B. McKay
null string, see Null string			
testing math-style (display, script or scriptscript)	II:2	46	B. McKay
Index Production	I:1	Appendix A	T. Winograd, W. Paxton
	II:2	A-28	M. Díaz
Letters	II:2	A-32–35	M. Díaz
Letterhead	II:2	A-33	M. Díaz
Lists	II:1	59, 72–72	A. Keller
	II:1	98–110	L. Price
	II:2	A-15	M. Díaz
Margins	II:2	A-19	M. Díaz
Matrices	II:2	A-30	M. Díaz
Memos	II:2	A-32–35	M. Díaz
Miscellaneous			
avoiding "Argument of (control sequence) has an extra }."	II:2	50	M. Spivak
conditional evaluation of macros	II:2	50	M. Spivak
\input within \if	II:2	50	M. Spivak
single tokens, identifying	II:2	52	M. Spivak
Multiplication	II:2	47	B. McKay

Nofill			
macros	II:1	59–60, 74–76	A. Keller
	II:2	A-16–18, 36	M. Díaz
program (SAIL)	II:1	87–93	L. Price, P. Milligan
program (Pascal)	II:1	94–97	L. Price, P. Milligan
program errata (SAIL and Pascal)	II:2	43–44	
Notes			
output to the writer on a separate file	II:1	60, 76, 85	A. Keller
printed at end of document	II:2	A-25	M. Díaz
Null string, testing for	II:1	60, 77	A. Keller
	II:2	51–52	M. Spivak
Numbering, page	II:1	57, 70–71	A. Keller
Output routines	II:1	57–58, 60–62, 71, 82–85	A. Keller
	II:2	A-18, 40	M. Díaz
Overlining	II:2	A-13	M. Díaz
Page numbering	II:1	57, 70–71	A. Keller
	II:2	A-18, 23	M. Díaz
Paragraphs			
beginning with large capital letters	II:1	60, 78	A. Keller
	II:2	A-16	M. Díaz
indented	II:1	58, 72	A. Keller
	II:2	A-13–15	M. Díaz
numbered, see Lists			
Parentheses, assorted sizes	II:2	A-11	M. Díaz
Pictures, plotting	II:2	48–49	B. McKay
Point, declaring font families of a particular ∼ size	II:1	56–57, 65–66	A. Keller
	II:2	A-11	M. Díaz
Proofs	II:2	A-31–32	M. Díaz
Recursion	II:2	46–48	B. McKay
	II:2	53	M. Spivak
References	II:2	A-25	M. Díaz
Roman numerals, uppercase	II:1	120–121	P. Milligan, L. Price
Spanish	II:2	A-12	M. Díaz
Table of Contents	II:1	60, 62, 86	A. Keller
	II:1	111–118	L. Price
	II:2	A-27–28	M. Díaz
Tables	II:2	A-25–27	M. Díaz
Testing			
math-style (display, script or scriptscript)	II:2	46	B. McKay
integral values	II:1	119–120	P. Milligan, L. Price
Theorems	II:2	A-31–32	M. Díaz
Top, baseline set to ∼ of box	II:1	60, 77	A. Keller
TUGboat submissions	II:1	53–54	B. Beeton
Underlining	II:1	59, 73	A. Keller
	II:2	A-13	M. Díaz
Uppercase letters			
large ∼ at beginning of paragraph	II:1	60, 78	A. Keller
	II:2	A-16	M. Díaz
Roman numerals	II:1	120–121	P. Milligan, L. Price
Verbatim			
mode	II:1	59–60, 74–76	A. Keller
	II:2	A-16–18, 36	M. Díaz
program (SAIL)	II:1	87–93	L. Price, P. Milligan
program (Pascal)	II:1	94–97	L. Price, P. Milligan

* * * * * * * * * * *

BUBBLES: A TEXTENSION IN SEARCH OF A TEXPERT

Timothy Murphy
Trinity College Dublin

Preamble

The bubble notation used in particle physics is identical in format with a notation for tensors due to the relativist Roger Penrose. This "language" is strictly formalised, with the possible diagrams governed by a few simple rules. In effect, a diagram consists of a number of "boxes", joined together by lines. A TEX extension is proposed, in which Penrose or bubble diagrams can be displayed in exactly the same way as mathematical formulae.

Analysis of the notation

There are 2 elements in the diagrams.

First, there are the **tensors** or **bubbles**. Each of these consists of a "container" of some kind, usually a rectangle or a circle. Inside there may be an identifying name or symbol.

Emerging from the perimeters of these containers are the second element of the diagrams, the **arms**. An arm may join 2 tensors; or it may extend to the edge of the diagram.

The relative positions of the arms on a tensor are significant, e.g. if the tensor T is represented by a rectangle, with 2 upper arms and 1 lower, then the left upper arm must be distinguished from the right upper arm; and both are quite different from the lower arm.

(If only that sought-for TEXpert could tell me how to replace this pedantic description of a box by a magical control sequence ... !)

Meaning of the Penrose notation

Although not strictly necessary, it may help if I explain, very briefly, the interpretation of the Penrose notation for tensors.

In the classical (Einstein) notation, a tensor T of type $(1,2)$ (for example) is denoted by

$$T^i_{jk}$$

Here i, j and k are "dummy suffixes", so that e.g.

$$T^a_{bc}$$

represents exactly the same tensor.

In the Penrose notation, T is incarcerated in a box, with 1 upper arm (corresponding to the upper index i) rising from the top of the box, and 2 lower arms (corresponding to the lower indices j and k) descending from the bottom of the box.

The joining of arms on 2 tensors (or on the same tensor) in a Penrose diagram corresponds to the **contraction** of the corresponding indices—denoted in the Einstein notation simply by repetition of the index. For example, the tensor

$$S^i_j T^k_{il},$$

obtained by contracting the upper index of S with an upper index of T, is represented by a diagram with 2 boxes, one for S with a single upper and a single lower arm, and one for T as above. The upper arm of S is joined to the left-hand lower arm of T, to represent the contraction. The remaining arms extend upwards or downwards (as the case may be) to the edge of the diagram. Thus the resulting tensor is of type $(1,2)$, with 1 upper arm, corresponding to k, and 2 lower arms, corresponding to j and l.

The relative positions of S and T are immaterial. They may be side-by-side; or S may be above T, or vice versa. The choice is made by the author on aesthetic or other grounds, e.g. to minimise the entanglement of arms. In this case, for example, the natural solution would be to place S above T.

Proposed macro definition

We shall use the control sequence \T for rectangular tensors with upper and lower arms.

The control sequence \T has 3 parameters: First, the name or symbol (possibly null) of the tensor. Next the upper arms, followed by a comma, and then the lower arms, e.g.

```
\T S i,j    \T T i,jk .
```

In other words, the definition starts

```
\.def\T#1#2,#3 {...}.
```

Turning to the 2ndand 3rdparameters, each of these consists of an 'hlist'. But the members of each list are in a strange new font, having the property that all letters in this font are represented by dots, of width (say) 1 em. The dots are positioned on the top and bottom edges of the box, and serve

(1) to determine the width of the box,
(2) to define the ends of the arms, and
(3) to label these arms.

Finally—and this is the difficult part—wherever a letter occurs twice in the parameter hlists of tensors, the corresponding points must be joined by lines. These joining arms must leave each tensor at right angles to the box—their paths being defined thereafter by cubic splines, a la METAFONT.

Other tensor shapes

We would like at least 2 other tensor formats.

First we should like to allow arms to emerge from all 4 edges of a rectangular box. This might be described by a control sequence

```
\q#1#2,#3,#4,#5
```

with the hlists corresponding to parameters 2 to 5 describing the arms emerging from the top, bottom, left and right of the box.

And we would also like to have circular tensors (or bubbles), with the property that arms could emerge in any radial direction, provided the cyclic order around the tensor was maintained. In general the arms would try to reach their destination as directly as possible. The control sequence for such tensors might be

```
\C#1#2
```

where #1 is the name of the tensor, as usual, while #2 is a single hlist giving the arms in (say) anticlockwise order.

Implementation

The first part of the construction, drawing the boxes, can be accomplished simply enough within TEX. We have not specified the horizontal or vertical spacing between boxes, but that is a mere detail.

The main problem arises, of course, when it comes to drawing the arms that link the boxes. Clearly that cannot be accommodated within standard TEX. However, the basic idea—joining 2 given points by a curve leaving the end-points in specified directions—is the very stuff of METAFONT.

The arms might need a gentle nudge, certainly, to circumnavigate the boxes, e.g. to determine on which side of a box to go when joining an upper arm to a lower on the same box.

But these are minor details. The essential question is: how can we save the hlists of "indices" appearing in the tensor parameter lists, and then join repeated indices by curved lines?

In conclusion

There have been several suggestions—some implemented—for interspersing TEX output with computerised graphics. It should be emphasised that our proposal is rather different from these.

We are asking for both more and less. More, in that we would like our bubbles to be device-independent in the same sense as TEX itself. Less, in that our graphical requirement is modesty itself—merely the ability to draw the crudest of boxes, and to join them with the simplest of curves.

Yet it may be that this kind of extension—the incorporation of graphical elements into the actual text—will prove more significant in the long run than the ability to draw the most beautiful of diagrams.

* * * * * * * * * * *

CHARTING YOUR GRAMMAR WITH TEX

Michael F. Plass
Xerox Corporation

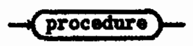
Are you one of those people who would rather look at a syntax chart than a BNF grammar? Do you avoid making a syntax chart for your language because it is too hard to draw or too hard to typeset? If so, this is the article for you. Pay attention, and you will learn how to use the macros below to create your own syntax charts with TEX.

First some basics. Every component of the syntax chart is enclosed in a TEX box, with the entry and exit points on the left and right sides of the box, aligned with the baseline. Usually the entry point is on the left end and the exit is on the right, but not always, as we shall see.

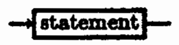
The simplest components are the boxes that represent the terminals and the nonterminals of the language. The terminals are the lowest level pieces of the language that the description deals with, for example, keywords, special characters, letters, digits, and so forth. Nonterminals are the names for the building blocks of the language. Some examples of nonterminal symbols might be number, identifier, expression, statement, program. If you are familiar with BNF, the nonterminals are the things in the angle brackets.

The terminals in a syntax chart are enclosed in boxes with rounded ends, and nonterminals are enclosed in rectangular boxes. This is how you use the syntax chart macros to make both kinds of elementary boxes:

```
\Terminal{procedure}      ⇒
```



```
\Nonterminal{statement}   ⇒
```

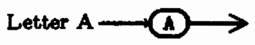


(Keep in mind that the case of the first letter in a TEX control sequence is significant.) You can control what fonts are used inside these boxes by defining the control sequences `\TerminalFont` and `\NonterminalFont` to be the appropriate font selectors. Notice each of these basic boxes have 'stems' on the left and right sides, one of them being an arrow and the other one just a line. TEX will determine which direction to point the arrow in on the basis of how the box is nested inside of other constructions.

The definition of a nonterminal is specified like this:

```
\Define{Letter A}\Terminal{A}\EndDef
```

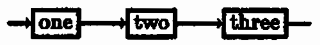
⇒



More complicated charts are built up by means of *sequencing*, *alternation*, and *repetition* of simpler charts. To illustrate the way these work, we will assume that the control sequence `\one` has been defined to be `\Nonterminal{one}`, and so forth.

Sequencing is easy to do; just put the subcharts together:

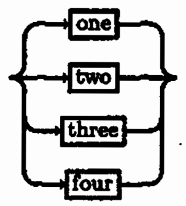
```
\one\two\three   ⇒
```



Alternation is a bit more complicated to specify. Here is an example:

```
\Alternatives
 {\Upper{\one}
  \Middle{\two}
  \Lower{\three}
  \Lower{\four}}
```

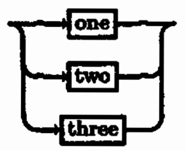
⇒



There may be any positive number of choices listed as the argument to `\Alternatives`, and each choice is marked by enclosing it in braces and preceding it with `\Upper`, `\Middle`, or `\Lower`. Things marked with `\Upper` go above the baseline, things marked with `\Lower` go below the baseline, and something marked with `\Middle` goes on the baseline. It is permitted to omit any one of these three kinds of tags:

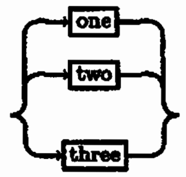
```
\Alternatives
 {\Middle{\one}
  \Lower{\two}
  \Lower{\three}}
```

⇒



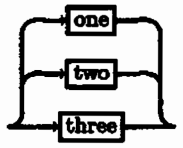
```
\Alternatives
 {\Upper{\one}
  \Upper{\two}
  \Lower{\three}}
```

⇒



```
\Alternatives
 {\Upper{\one}
  \Upper{\two}
  \Middle{\three}}
```

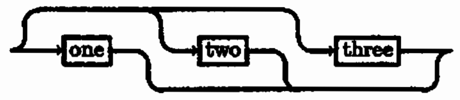
⇒



Sometimes a stack of alternatives can get very tall; in this case it might be better to spread them out horizontally:

```
\HorzAlternatives{\Alternative{\one}
 \Alternative{\two}\Alternative{\three}}
```

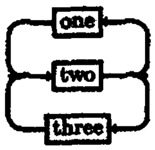
⇒



This one is especially appropriate for a long list of short choices.

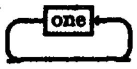
Repetition is specified in almost exactly the same way as alternatives:

```
\Repeat{\Upper{\one}
        \Middle{\two}     ⇒
        \Lower{\three}}
```

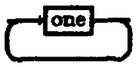


The same rules apply as before, except that you are not allowed to leave out the `\Middle`. This is not the same as specifing an empty middle, which you might often want to do:

```
\Repeat{\Upper{\one}\Middle{}}   ⇒
```



```
\Repeat{\Middle{\one}\Lower{}}   ⇒
```



Sometimes a chart gets too wide for the page, and TEX breaks it up into lines as if it were a paragraph. If this is want you want, fine—but if you are in unrestricted horizontal mode, you can make a nicer transition to a new line by using `\NewLine`. Look at Appendix B for examples of this.

Now you know about all of the constructions. They can be nested in any way you please, up to a depth of four. One way to get around this limit is to save away the innermost parts of the diagram in a box and call it in when you need it. You can say `\AllocBox\cntrlseq` to define `\cntrlseq` to be a digit indicating a box that you can use; this way your box usage won't conflict with that of the syntax chart macros. Remember to use braces to enclose the section in which you use `\AllocBox`, so that the box number can be used again. If you exceed TEX's `varsize` limit, the usual solution is to force a page eject so TEX does not have to keep so much stuff in its memory.

To use the macros, say

```
\input SynChart
```

near the beginning of your file. When you actually want to make a chart, say `\SyntaxChart` to set up the baselineskip, lineskip, and some other stuff. If you are mixing syntax charts with other text, you will want to enclose the `\SyntaxChart` along with the chart definition in braces, so all the funny definitions go away at the end of the group and don't mess up the rest of your document. You will probably also want to say `\parindent 0pt` to eliminate the paragraph indentation.

The call on `\SyntaxChart` also tells TEX to ignore tabs and carriage returns so you can format your more easily. Just be careful where you add spaces—after a control sequence is OK, but not between or after the parameters to a macro. The reason for not ignoring spaces is so terminals and nonterminals can have embedded spaces; if you prefer, you can say `\IgnoreWhiteSpace` to cause spaces to be ignored too, and then put in a control-space where you really want a space.

Now would be a good time to go try making your own simple syntax chart, using the macro definitions in Appendix A. When defining a complex diagram, it is often helpful to first define pieces of it as TEX macros. This makes the source much easier to read, and keeps the nesting of braces down to a reasonable level. This device is used extensively in the example in Appendix B.

Finally, some fine points about spacing. The horizontal lines are actually composed of fixed-width rules and leaders filled with rules. The control sequence `\QLine` defines a fairly short fixed-width line, and `\QQLine` defines one twice as long. `\Fil` defines a line that stretches like glue, with a normal width of zero and the same stretchability as `\hfil`; `\Fill` is similar, but with the stretchability of an `\hfill`. Each component of a repetition or (vertical) alternation construction implicitly has `\Fil` on each side, so if the components are not all the same width, the available space in each of the shorter ones will be distributed evenly between the embedded `\Fil`s and the ends of the component. By using these macros in various combinations, you can get any kind of horizontal spacing you want.

When a terminal box contains only special character or only upper case letters, the result looks best when the contents are centered vertically in the box; this is the default. However, if the same thing is done for lower case letters, the baselines of the words in different boxes may not line up due to the pattern of ascenders and descenders. The way to fix this is by using a *strut*, which is a zero-width box whose height and depth match the extremes of the font. If the keywords in your language are in lower case, say `\Strut` after the keyword in the terminal box—or, better yet, define a macro that does this, as in Appendix B. Struts are already included in nonterminal boxes, because these contain mostly lower-case letters.

Sometimes two rows would look better if they were moved a little bit closer together. You can do this by using `\TrimTop` or `\TrimBot`; these macros take an hlist as a parameter, box it, and then take a little off of the top or the bottom. Look at Appendix B to see how these can be used.

Appendix A. Listing of SynChart.TEX

```
%%%%%% Beginning of SynChart.TEX

% editor's note - fonts have been adjusted to conform to TUGboat usage;
% the original values are retained, commented out
\font G=dragon10                        % \font >=MANFNT
%                                          \font U=CMTT8
%                                          \font c=CMR8

% The following macros declare how white space is treated.
\def\IgnoreLinebreaks{\chcode'15=9\chcode'11=9}
\def\IgnoreWhiteSpace{\chcode'15=9\chcode'11=9\chcode'40=9}
\def\DontIgnoreWhiteSpace{\chcode'15=5\chcode'11=10\chcode'40=10}

\IgnoreWhiteSpace % so the macro definitions may be freely formatted.

\def\NonterminalFont{\:c}        % font to use for nonterminal symbols
\def\TerminalFont{\:U}                % font to use for terminal symbols
% \def\GraphicFont{\:>}                % font to get quarter circles from;
\def\GraphicFont{\:G}               % font to get quarter circles from;
                                % should have characters a, b, c, and d like
                                % the DRAGON font in the metafont manual.

% The \LeftArrow, \RightArrow, and \ArrowLine macros define the stems
% that get put on each terminal or nonterminal box.  The appropriate arrow
% is put on the entry side of each box, and \ArrowLine is put on the exit side
% to balance out the arrow.  These definitions supply a rather small arrow
% with a long stem, using characters from the standard CMSY10 font.  Users
% that have good arrowheads available in other fonts should redefine these
% macros to take advantage of them.
\def\LeftArrow {\hskip 1.2pt
  \Strikeout{\vbox to 1.9pt{\hbox{\hskip-2pt\:u\char'40\hskip-0.5pt}\vss}}}
\def\RightArrow
 {\Strikeout{\vbox to 1.9pt{\hbox{\hskip-0.5pt\:u\char'41\hskip-2pt}
                           \vss}}\hskip 1.2pt}
\def\ArrowLine{\Strikeout{\hskip 8.7pt}}

% The next two macros define big arrowheads.
\def\BigLeftArrow{\Strikeout{\CenterSink{\hbox{\hskip-2vu\bf<}}}}
\def\BigRightArrow{\Strikeout{\CenterSink{\hbox{\bf>\hskip-2vu}}}}

% The next few macros describe the dimensions of the quarter-circle font.
\def\QThickness{1.1pt} % Thickness of strokes
\def\QSize{5pt} % Height and width of the quarter circles
\def\QQSize{10pt} % Twice \QSize
\varunit\QThickness % 1vu is the stroke thickness.

% The next two groups of macros may need editing if a font other than the
% dragon font is used to get the quarter circles.

% \I, \II, \III, \IV are the quarter circles in the first, second, third and
% fourth quadrants, numbered counterclockwise starting with the upper-right one.
\def\I{\lower \QSize \hbox{\GraphicFont a\BackQ}}
\def\II{\lower \QSize \hbox{\BackQ\GraphicFont d}}
\def\III{\lower \QSize \hbox{\BackQ\GraphicFont c}}
\def\IV{\lower \QSize \hbox{\GraphicFont b\BackQ}}

% \LeftRound and \RightRound are the end caps for Terminal boxes
\def\LeftRound {\BackQ\hbox{\GraphicFont d}\BackQ\BackQ
                \lower\QQSize\hbox{\GraphicFont c}}
\def\RightRound{\hbox{\GraphicFont a}\BackQ\BackQ
                \lower\QQSize\hbox{\GraphicFont b}\BackQ}

% editor's note - page break for TUGboat publication
% \LeftSquare and \RightSquare are the end caps for Nonterminal boxes
\def\LeftSquare
 {\vrule height\QQSize depth\QQSize width 0pt
  \vrule height\QSize depth\QSize width 1vu \hskip -1vu}
\def\RightSquare
 {\hskip -1vu \vrule height\QSize depth\QSize width 1vu}

% These macros are for moving forward and backward in Q units.
\def\BackQ{\hskip-\QSize}
\def\BackQQ{\hskip-\QQSize}
\def\ForwQ{\hskip \QSize}
\def\ForwQQ{\hskip \QQSize}

% Macros in this group are meant to be used directly by the user.
\def\Empty{\vrule height\QSize depth\QSize width 0pt}
\def\Strut{\save0\hbox{Bg}\vrule height 1ht0 depth 1dp0 width 0pt}
\def\QLine{\vrule height 0.5vu depth 0.5vu width \QSize}
\def\QQLine{\vrule height 0.5vu depth 0.5vu width \QQSize}
\def\Fil{\leaders\HorzLine\hfil}
\def\Fill{\leaders\HorzLine\hfill} % use this to swamp a \Fil
\def\FollowedBy{\Fil} % use to separate symbols in a sequence
\def\TrimTop#1{\vbox expand -\Qsize{\vss\hbox{#1}}}
\def\TrimBot#1{\hbox{$\vtop{\hbox{#1}\vskip-\QQSize}$}}

% Some useful formatting primitives, used internally.
\def\HorzLine{\hrule height 0.5vu depth 0.5vu}
\def\Strikeout
  #1{\save0\hbox{#1}\hbox to 1wd0
        {\vrule height 0.5vu depth 0.5vu width 1wd0\hss\unbox0}}
\def\CenterSink#1{\lower\QSize\vbox to \QQSize{\vss\hbox{#1}\vss\null}}
\def\Sandwch#1{\lower\QSize
               \vbox to \QQSize
                {\vskip-0.5vu\HorzLine\vss
                \hbox{#1}\vss
                \HorzLine\vskip-0.5vu\null}}
\def\VertLine#1{\hskip -0.5vu \vrule #1 width 1vu \hskip -0.5vu}
\def\LinetoTop{\VertLine{depth-\QSize}}
\def\LinetoBot{\VertLine{height-\QSize}}
\def\FilVertLine{\hskip -0.5vu \vrule width 1vu \hskip -0.5vu}

% These define some common bits and pieces of the diagrams.
\def\SwitchUp{\IV\BackQ\QQLine}
\def\SwitchDn{\I\BackQ\QQLine}
\def\MergeUp{\QQLine\BackQ\II}
\def\MergeDn{\QQLine\BackQ\III}

% Use \Define to start a section of the syntax diagram.
\def\Define#1{\par${}$\CenterSink
  {\NonterminalFont\Strut#1}\hskip 2vu plus 300pt
  \penalty 0\QQLine}

% And use \EndDef to end it.
\def\EndDef{\QLine\BigRightArrow\par}

% If a section gets too wide, \NewLine will continue the chart on a new line.
\def\NewLine
   {\Fil\I\LinetoBot\ForwQQ\linebreak
    \hbox{\ForwQQ\ForwQQ}\LinetoBot\II
    \Fil\vrule height \QQSize depth \QQSize width 0pt\BigLeftArrow\Fil
    \IV\LinetoTop\ForwQQ\linebreak
    \hbox{\ForwQQ\ForwQQ}\LinetoTop\III\Fil}

% editor's note - page break for TUGboat publication
% The macros \LeftToRight, \RightToLeft, and \SwitchDirection control the
% way arrows are pasted onto the low-level boxes of the chart.
\def\LeftToRight
 {\def\LeftSideArrow{\RightArrow}
  \def\RightSideArrow{\ArrowLine}
  \def\SwitchDirection{\RightToLeft}}
\def\RightToLeft
 {\def\LeftSideArrow{\ArrowLine}
  \def\RightSideArrow{\LeftArrow}
  \def\SwitchDirection{\LeftToRight}}

% The next two macros are used for creating terminal and nonterminal boxes.
\def\Nonterminal
 #1{\LeftSideArrow\LeftSquare\Sandwch
    {\NonterminalFont
     \ #1\Strut\ }\RightSquare\RightSideArrow}
\def\Terminal
 #1{\LeftSideArrow\LeftRound\Sandwch
    {\TerminalFont
     #1}\RightRound\RightSideArrow}

% The next group of macros are used for allocating digits for referring to
% counters and boxes.  These are needed only because counters and boxes in TEX
% do not nest in the nice way that macro definitions do.  The macros are a
% bit on the tricky side, but are short enough that you can figure out how
% they work if you understand the way TEX expands macros.  If you are using
% other packages that use some of these boxes or counters, you may have to
% change some of the numbers in these macros.
\def\Alloc#1#2{\def#2{#1}}
\def\AllocBox
 {\def\AllocBox{\def\AllocBox{\def\AllocBox{\def\AllocBox
 {\def\AllocBox{\def\AllocBox{\def\AllocBox{\def\AllocBox
 {\def\AllocBox{\Overflow
  }\Alloc9}\Alloc8}\Alloc7}\Alloc6}\Alloc5}\Alloc4}\Alloc3}\Alloc2}\Alloc1}
\def\AllocCtr
 {\def\AllocCtr{\def\AllocCtr{\def\AllocCtr{\def\AllocCtr
  {\Overflow}\Alloc8}\Alloc7}\Alloc6}\Alloc5}


% Here're the ones you've been waiting for.  Both \Alternatives and
% \Repeat work in roughly the same fashion, so this description applies
% tc them both.  The single argument is composed of the subcomponents,
% each one enclosed in braces and preceded by a control sequence.  These
% control sequences get defined as local macros, and grab the subcomponents
% as parameters.  Actually, this happens twice: the first time the argument
% is interpreted, the local macros just ignore their parameter, and the
% second time through, the real work is done.  The first pass is used to
% count the subcomponents, since the last one has to be treated specially,
% and also does some validation of the arguments.
%
% There are places in these macros where a \gdef or \xdef is used to get some
% information out of a local scope or to force expansion of a count; since
% the value is used before any other macros are expanded, there is no danger
% of nested macro calls messing things up.
%
% If the structure of the macro isn't tricky enough for you, you can try to
% understand the way the result is built up out of boxes.  Here is the basic
% idea: while the stuff in the middle is being built up in an \halign, the
% connectors on the side are built up in two boxes, one for the left and the
% other on the right.  The baselines of the boxes on the sides are maintained
% to be correct for the final result by doing a \vbox above and at the middle
% and a \vtop after the middle.  The baseline of the stuff in the middle is
% ignored, and when it is all built, it is boxed and forced to line up with
% the boxes on either side.  The \halign is needed since each row has to be
% expanded to the maximum width of everything in the middle.
\def\Alternatives
    #1{{\AllocCtr\AltCtr
```

```
        \setcount\AltCtr 0
        \def\state{T}
        \def\Upper##1{\if L\state{\BadUseOfUpper}\else{}
                      \advcount\AltCtr\def\state{U}}
        \def\Middle##1{\if L\state{\BadUseOfMiddle}\else{}
                       \advcount\AltCtr\def\state{L}}
        \def\Lower##1{\advcount\AltCtr\def\state{L}}
        #1
        \if L\state{}\else{\NoMiddle}
        \def\state{T}
        \def\Upper{\advcount\AltCtr by -1
                   \UpperAlternative}
        \def\Middle{\advcount\AltCtr by -1
                    \MiddleAlternative}
        \def\Lower{\if L\state {\gdef\OMA{}}
                   \else {\gdef\OMA{\OmittedMiddleAlternative}}
                   \OMA % Needed because of scoping in the if statement
                   \advcount\AltCtr by -1
                   \LowerAlternative}
        \AllocBox\L\save\L\null
        \AllocBox\R\save\R\null
        \save0\hbox{$\vtop{\null\halign{##\cr#1\null\cr}}$}
        \hbox{\ForwQ\box\L\raise 1ht\R\box0\box\R\ForwQ}}}

% One extra tricky thing about \Repeat is that the scope of \SwitchDirection
% is terminated by the \cr at the end of the alternatives.  Maybe you didn't
% know about this scoping rule; it was sure news to me.
\def\Repeat
    #1{{\AllocCtr\AltCtr
        \setcount\AltCtr 0
        \def\state{T}
        \def\Upper
            ##1{\if L\state{\BadUseOfUpper}\else{}
                \advcount\AltCtr\def\state{U}}
        \def\Middle
            ##1{\if L\state{\BadUseOfMiddle}\else{}
                \advcount\AltCtr\def\state{L}}
        \def\Lower
            ##1{\if L\state{}\else{\BadUseOfLower}
                \advcount\AltCtr\def\state{L}}
        #1
        \if L\state{}\else{\NoMiddle}
        \def\state{T}
        \def\Upper{\SwitchDirection\advcount\AltCtr by -1
                   \UpperAlternative}
        \def\Middle{\advcount\AltCtr by -1
                    \RepeatBody}
        \def\Lower{\SwitchDirection\advcount\AltCtr by -1
                   \LowerAlternative}
        \AllocBox\L\save\L\null
        \AllocBox\R\save\R\null
        \save0\hbox{$\vtop{\null\halign{##\cr#1\null\cr}}$}
        \hbox{\box\L\raise 1ht\R\box0\box\R}}}

% editor's note - page break for TUGboat publication
```

```
\def\UpperAlternative
 #1{\save0\hbox{#1\Empty}
    \save\L\vbox{\box\L\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                                 \if U\state{\FilVertLine}\else{\LinetoBot}\II}}
    \save\R\vbox{\box\R\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                                 \I\if U\state{\FilVertLine}\else{\LinetoBot}}}
    \Fil\unbox0\Fil\cr\def\state{U}}

\def\MiddleAlternative
 #1{\save0\hbox{#1\Empty}
    \xdef\RowsToGo{\count\AltCtr}
    \save\L\vbox{\box\L\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \if U\state{\BackQ\IV\LinetoTop}\else{}
                             \if 0\RowsToGo{}\else{\BackQ\I\LinetoBot}
                             \BackQ\QQLine}}
    \save\R\vbox{\box\R\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \QQLine\BackQ
                             \if U\state{\LinetoTop\III\BackQ}\else{}
                             \if 0\RowsToGo{}\else{\LinetoBot\II\BackQ}}}
    \Fil\unbox0\Fil\cr\def\state{L}}

\def\OmittedMiddleAlternative
   {\save0\hbox{\Empty}
    \xdef\RowsToGo{\count\AltCtr}
    \save\L\vbox{\box\L\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \if U\state{\BackQ\IV\LinetoTop}\else{}
                             \if 0\RowsToGo{}\else{\BackQ\I\LinetoBot}
                             \ForwQ}}
    \save\R\vbox{\box\R\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \ForwQ
                             \if U\state{\LinetoTop\III\BackQ}\else{}
                             \if 0\RowsToGo{}\else{\LinetoBot\II\BackQ}}}
    \hfil\Empty\cr\def\state{L}}

\def\RepeatBody
 #1{\save0\hbox{#1\Empty}
    \xdef\RowsToGo{\count\AltCtr}
    \save\L\vbox{\box\L\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \if U\state{\LinetoTop\III\BackQ}\else{}
                             \if 0\RowsToGo{}\else{\LinetoBot\II\BackQ}
                             \QLine}}
    \save\R\vbox{\box\R\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
                             \QLine
                             \if U\state{\BackQ\IV\LinetoTop}\else{}
                             \if 0\RowsToGo{}\else{\BackQ\I\LinetoBot}}}
    \Fil\unbox0\Fil\cr\def\state{L}}

\def\LowerAlternative
 #1{\save0\hbox{#1}
    \xdef\RowsToGo{\count\AltCtr}
    \save\L\hbox{$\vtop{\box\L\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
            \if 0\RowsToGo {\LinetoTop}\else{\FilVertLine}\III}}$}
    \save\R\hbox{$\vtop{\box\R\hbox{\vrule height 1ht0 depth 1dp0 width 0pt
            \IV\if 0\RowsToGo{\LinetoTop}\else{\FilVertLine}}}$}
    \Fil\unbox0\Fil\Empty\cr\def\state{L}}

% editor's note - page break for TUGboat publication
% Horizontal alternatives work in much the same way as the others,
% but are a little simpler because we don't have to worry about a \Middle
% and we can just use an \hbox instead of \vboxing an \halign and fooling
% around with the position of the baseline.
\def\HorzAlternatives
    #1{{\AllocCtr\AltCtr \setcount\AltCtr 0
        \def\Alternative##1{\advcount\AltCtr}
        #1% count the number of alternatives
        \def\Alternative
          {\advcount\AltCtr by -1\xdef\AltsToGo{\count\AltCtr}
           \if 0\AltsToGo{\gdef\HAlt{\LastHorzAlternative}}\else
                           {\gdef\HAlt{\HorzAlternative}}
           \HAlt}
        \def\HorzAlternative
            {\def\HorzAlternative
                {\MiddleHorzAlternative}\FirstHorzAlternative}
        \AllocBox\TopTrack
        \AllocBox\BotTrack
        \save\TopTrack\null
        \save\BotTrack\null
        \save0\hbox{#1}
        \vbox{\box\TopTrack\hbox{$\vtop{\box0\box\BotTrack}$}}}}

\def\FirstHorzAlternative
 #1{\save0\hbox{\QLine#1\I}
    \save\TopTrack\hbox{\box\TopTrack\ForwQ\hbox to 1wd0{\II\Fil}}
    \save\BotTrack\hbox{\box\BotTrack\ForwQ\hskip 1wd0}
    \IV\BackQ\Qline\LinetoTop\box0\LinetoBot}

\def\MiddleHorzAlternative
 #1{\save0\hbox{\III#1\I}
    \save\TopTrack\hbox{\box\TopTrack\I\hbox to 1wd0{\BackQ\Fil}}
    \save\BotTrack\hbox{\box\BotTrack\III\hbox to 1wd0{\Fil\BackQ}}
    \ForwQ\LinetoTop\box0\LinetoBot}

\def\LastHorzAlternative
 #1{\save0\hbox{\III#1\QQLine\BackQ}
    \save\TopTrack\hbox{\box\TopTrack\I}
    \save\BotTrack\hbox{\box\BotTrack\III\hbox to 1wd0{\Fil\IV}}
    \ForwQ\LinetoTop\box0\LinetoBot\II}

\def\SyntaxChart
       {\par
        \varunit\QThickness
        \lineskip0pt
        \baselineskip-1pt
        \parfillskip 0pt plus 3000pt
        \parskip \QQsize plus \Qsize
        \IgnoreLinebreaks
        \LeftToRight}

\DontIgnoreWhiteSpace % so the user can space significantly

%%%%%% End of SynChart.TEX
```

Appendix B. Here is a non-trivial example of the use of the syntax chart macros.

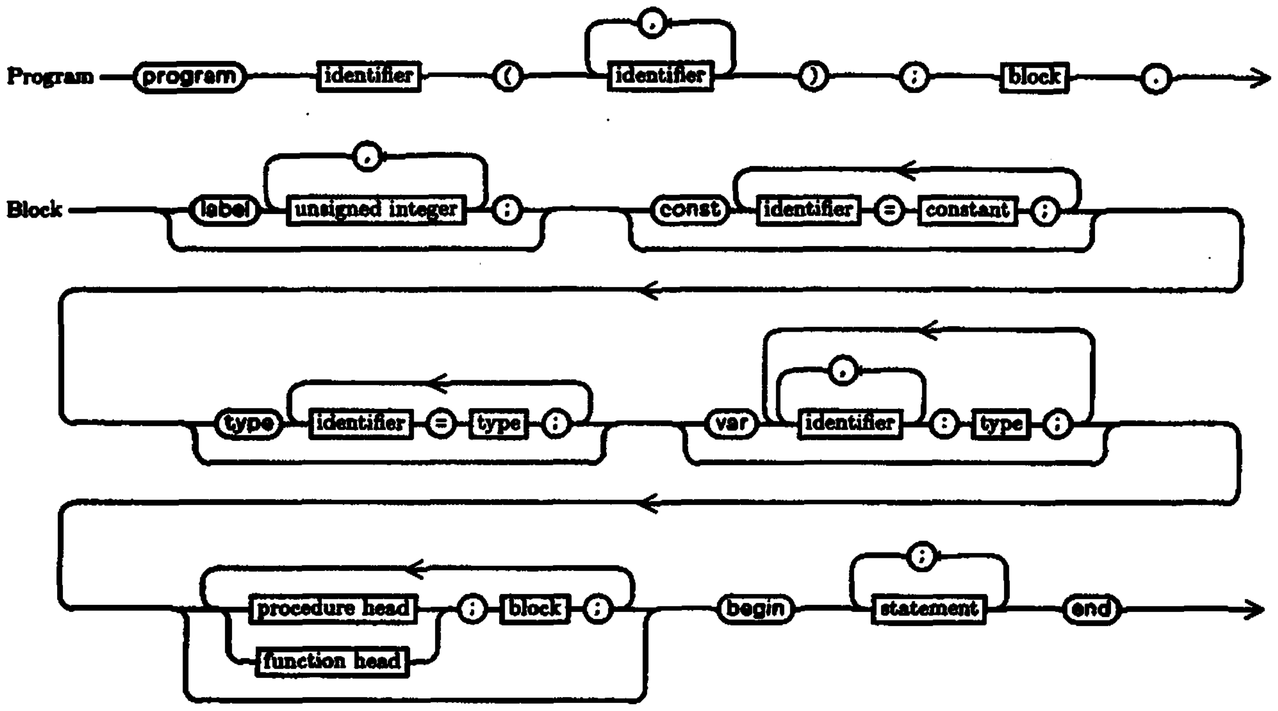
```
  %%%%%% Beginning of PascalSyntax.TEX
\font K=CMSS8 % for the keywords in the chart.

{\SyntaxChart
\AllocBox\subchartbox % for saving pieces of the diagrams

\parindent 0pt

%\def\LeftArrow{\vrule height 0.5vu depth 0.5vu width 2.5vu}
% Turn off the right arrows because they make the charts too wide.
\def\RightArrow{\vrule height 0.5vu depth 0.5vu width 2.5vu}
\def\ArrowLine{\vrule height 0.5vu depth 0.5vu width 2.5vu}
\def\NoArrows{\def\LeftArrow{}\def\RightArrow{}\def\ArrowLine{}}

% \T{<char>} yields a circular terminal node with the character
% aesthetically placed, assuming CMTT8 is used. The 4.5417pt is the
% height of plus and minus, and 4.2pt is the character width.
% Colons and periods should be followed by a \CommaStrut to make
% them line up with semicolons and commas.  For parens, brackets,
% and braces, put an extra \hss on the outer side to center them better.
% This macro may be used for multi-character terminal symbols, but
% \KW should be used for keywords.
\def\T#1{\Terminal
        {\vbox to 4.5417pt{\vss\hbox expand -4.2pt{\hss#1\hss}\vss\vfilneg}}}

\def\CommaStrut{\save0\hbox{,}\lower 1dp0\null}

\def\KW#1{\Terminal{\:K \hskip-2pt#1\Strut\hskip-2pt}} % Use this for keywords.

\def\N{\Nonterminal} % just an abbreviation.

\def\Optional#1{\Alternatives{\Middle{#1}\Lower{\Empty}}}
\def\Rept#1{\Repeat{\Upper{\BigLeftArrow}\Middle{#1}}}
\def\Star#1{\Repeat{\Upper{#1}\Middle{\Empty}}}
\def\LongStar#1{\Optional{\Rept{#1}}}
\def\TwoAlts[#1|#2]{\Alternatives
        {\Upper{\TrimBot{#1}}\Lower{\TrimTop{#2}}}}
\def\RAW#1#2{\Repeat{\Upper{#1}\Middle{#2}}} % Repeat Above With
\def\RBW#1#2{\Repeat{\Middle{#2}\Lower{#1}}} % Repeat Below With
\def\\{\Fil}
\def\-{\QQLine}

\def\goodbreak{\vfil\null\vfilneg}

\def\singlequote{\char'177} % symmetric single quote in tty font.

\def\lpar{\T{\hss(}} \def\rpar{\T{)\hss}}
\def\lbrak{\T{\hss[}} \def\rbrak{\T{]\hss}}
\def\comma{\T{,}}
\def\colon{\T{:\CommaStrut}}
\def\eq{\T{=}}
\def\semicolon{\T{;}}
\def\period{\T{.\CommaStrut}}
\def\gets{\T{\save0\hbox{>}\vbox
               {\hbox{\CenterSink{:}\CenterSink{\vbox to 1ht0{}=}}\null}}}

% editor's note - page break for TUGboat publication
```

```
\def\constant{\N{constant}}
\def\type{\N{type}}
\def\statement{\N{statement}}
\def\id{\N{identifier}}
\def\idlist{\RAW\comma\id}
\def\expression{\N{expression}}
\def\variable{\N{variable}}
\def\character
 {\Alternatives
        {\Middle{\N{character}}
         \Lower{\T{\singlequote\singlequote}}}}
\def\digits{\Rept{\N{digit}}}
\def\blockbody{\KW{begin}\\\RAW\semicolon\statement\\\KW{end}}
\def\sign
{\Alternatives
        {\Upper{\T{+}}
         \Middle{}
         \Lower{\T{-}}}}

\Define{Program}
\KW{program}\\\id\\\lpar\\\RAW\comma\id\\\rpar\\\semicolon\\\N{block}\\\period
\Fil\EndDef

{ % Block
\def\labelpart
 {\Optional{\KW{label}\\\RAW\comma{\N{unsigned integer}}\\\semicolon}}

\def\constpart
 {\Optional{\KW{const}\\\Rept{\id\\\eq\\\constant\\\semicolon}}}

\def\typepart
 {\Optional{\KW{type}\\\Rept{\id\\\eq\\\type\\\semicolon}}}

\def\varpart
 {\Optional{\KW{var}\\\Rept{\idlist\\\colon\\\type\\\semicolon}}}

\def\prochead{\N{procedure head}}
\def\funchead{\N{function head}}
\def\pfhead{\Alternatives{\Middle{\prochead}\Lower{\funchead}}}

\def\pfdeclpart{\LongStar{\pfhead\\\semicolon\\\N{block}\\\semicolon}}


\Define{Block}
\Fil\labelpart\\\constpart\Fil\NewLine
\Fil\typepart\\\varpart\Fil\NewLine
\Fil\pfdeclpart\\\blockbody\Fil\Fil\EndDef

} % end of Block

\vfil\eject
% editor's note - page break for TUGboat publication
```

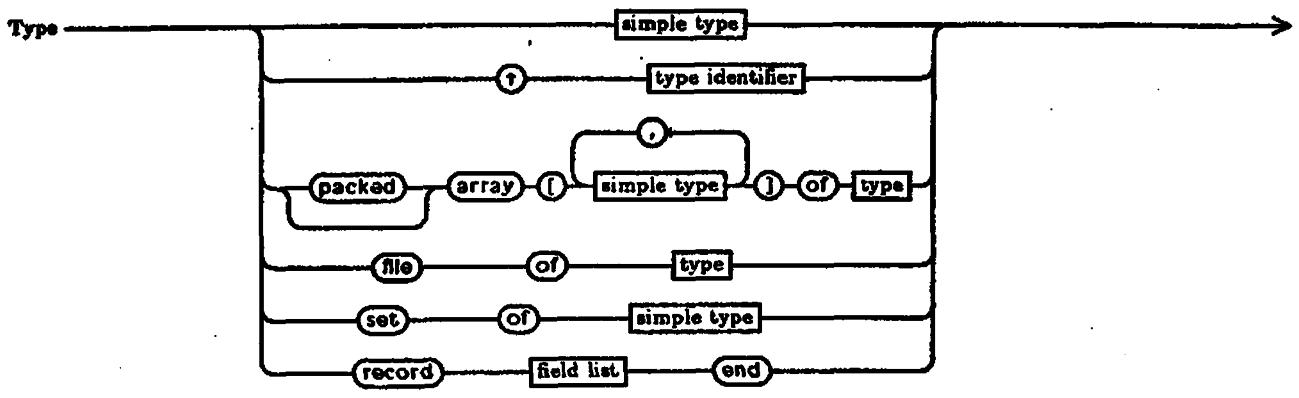
```
{ % Declaration Extras
\def\simpletype{\N{simple type}}
\def\typelist{\lbrak\\\RAW\comma{\\\simpletype\\}\\\rbrak}
\Define{Type}\Fil
\Alternatives
       {\Middle{\Fil\simpletype}
        \Lower{\Fil\T{\char'136}\\\N{type identifier}}
        \Lower{\Optional{\KW{packed}}\\\KW{array}\\\typelist\\\KW{of}\\\type}
        \Lower{\KW{file}\\\KW{of}\\\type\\}
        \Lower{\KW{set}\\\KW{of}\\\simpletype\\}
        \Lower{\KW{record}\\\N{field list}\\\KW{end}\\}}
\Fil\Fil\EndDef

\Define{Simple type}\Fil
\Alternatives
       {\Middle{\Fil\N{type identifier}}
        \Lower{\Fil\lpar\\\idlist\\\rpar}
        \Lower{\Fil\constant\\\T{..\CommaStrut}\\\constant}}
\Fil\Fil\EndDef

% \vfil\eject

\def\varianthead{\KW{case}\\\id\\\colon\\\N{type identifier}\\\KW{of}}
\def\variant{\RAW\comma\constant\\\colon\\\lpar\\\N{field list}\\\rpar}
\Define{Field list}\Fil
\LongStar{\Optional{\idlist\\\colon\\\type}\\\semicolon}\NewLine
\save\subchartbox\hbox{\Optional{\variant}}\!% this gets too complicated
\Optional{\varianthead\\\LongStar{\box\subchartbox\\\semicolon}}
\Fil\EndDef

\vfil\eject

\def\funorvar{\TwoAlts[\KW{function}|\KW{var}]}
\def\formalparmgroup
{\Alternatives
       {\Middle{\funorvar\\\idlist\\\colon\\\N{type identifier}}
        \Lower{\KW{procedure}\\\RBW\comma\id}}}
\Define{Parameter list}
\save\subchartbox\hbox{\formalparmgroup}\!% again too complicated
\Optional{\lpar\\\RAW\semicolon{\box\subchartbox}\\\rpar}
\Fil\EndDef

} % end of Declaration Extras
% \vfil\eject

{ % Statement
\def\varorfunid{\TwoAlts[\N{variable}|\N{function identifier}]}
\def\assignment{\varorfunid\\\gets\\\expression}

\def\procid{\N{procedure identifier}}
\def\actualarglist
   {\lpar\\\RAW\comma
     {\Alternatives
       {\Middle{\expression}
        \Lower{\procid}}}\\\rpar}
\def\proccall{\procid\\\Optional{\actualarglist}}

\def\ifstatement{\KW{if}\\\expression\\\KW{then}\\\statement
                 \\\Optional{\KW{else}\\\statement}}

\def\casepart{\RBW\comma\constant\\\colon\\\statement}
\def\cases{\Optional{\RAW\semicolon\casepart}}
\def\casestatement{\KW{case}\\\expression\\\KW{of}\\\cases\\\KW{end}}

\def\whilestatement{\KW{while}\\\expression\\\KW{do}\\\statement}

\def\repeatstatement{\KW{repeat}\\\RAW\semicolon\statement
```

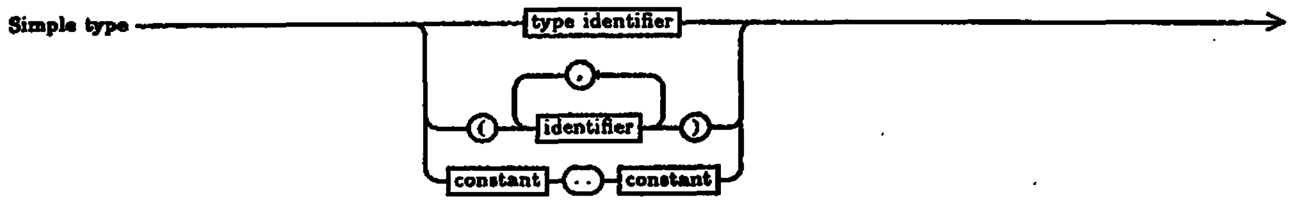
```
                        \\\KW{until}\\\expression}

\def\toordownto{\TwoAlts[\KW{to}|\KW{downto}]}
\def\forstatement{\KW{for}\\\N{variable identifier}\\\gets\\\expression
                  \\\toordownto\\\expression\\\KW{do}\\\statement}

\def\withstatement{\KW{with}\\\RAW\comma\variable\\\KW{do}\\\statement}

\def\gotostatement{\KW{goto}\\\N{unsigned integer}}


\Define{Statement}\Fil
\Optional{\N{unsigned integer}\\\colon}\\\NewLine
\Alternatives
       {\Middle{\assignment\\}
        \Lower{\proccall\\}
        \Lower{\blockbody\\\Fil}
        \Lower{\ifstatement}
        \Lower{\casestatement}
        \Lower{\whilestatement\\}
        \Lower{\repeatstatement\\}
        \Lower{\forstatement\\}
        \Lower{\withstatement\\}
        \Lower{\gotostatement\\\Fil\Fil}
        \Lower{\Empty}}
\Fil\EndDef
} % end of Statement

\vfil\eject

{ % Expression
\def\relop{{\Fil\NoArrows
     \HorzAlternatives
        {\Alternative{\T{=}}
         \Alternative{\T{<}}
         \Alternative{\T{>}}
         \Alternative{\T{<>}}
         \Alternative{\T{<=}}
         \Alternative{\T{>=}}
         \Alternative{\KW{in}}}}}
\Define{Expression}
\N{simple expression}\\
\Optional{\relop\\\N{simple expression}}
\Fil\EndDef

\Define{Simple expression}\Fil
\Alternatives{\Upper{\T{+}}\Middle{\Empty}\Lower{\T{-}}}\\
\Repeat{\Upper{\T{+}}
        \Upper{\T{-}}
        \Middle{\N{term}}
        \Lower{\KW{or}}}
\Fil\Fil\Fil\Fil\EndDef

\def\mulop{{\NoArrows
    \HorzAlternatives
       {\Alternative{\T{*}}
        \Alternative{\T{/}}
        \Alternative{\KW{div}}
        \Alternative{\KW{mod}}
        \Alternative{\KW{and}}}}}
\Define{Term}\Fil
\N{factor}\\\LongStar{\mulop\N{factor}}
\Fil\Fil\Fil\Fil\EndDef

% editor's note - page break for TUGboat publication
```

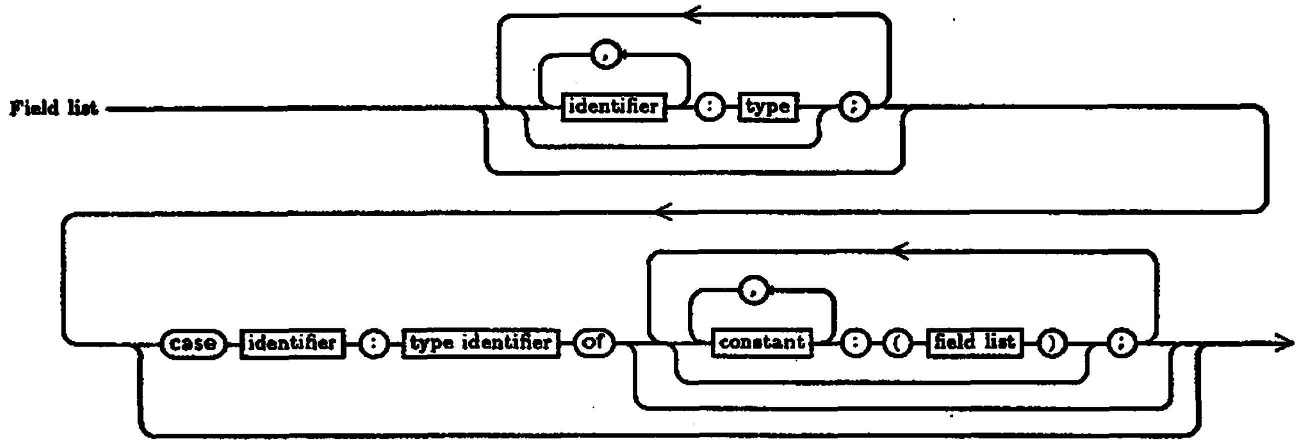
```
\def\actualarglist
       {\lpar\\\RAW\comma\expression\\\rpar}
\save\subchartbox\hbox
       {\lbrak
        \\\Optional{\RAW\comma
                         {\expression
                          \Optional{\T{..\CommaStrut}\\\!
                                     \expression}}}\\\!
        \rbrak}
\Define{Factor}\Fil
\Alternatives
       {\Middle{\N{unsigned constant}}
        \Lower{\variable}
        \Lower{\N{function identifier}\\\Optional\actualarglist}
        \Lower{\lpar\\\expression\\\rpar}
        \Lower{\KW{not}\\\N{factor}}
        \Lower{\box\subchartbox}}
\Fil\Fil\EndDef

\Define{Variable}\Fil
\TwoAlts[\N{variable identifier}|\N{field identifier}]\\
\Repeat{\Upper{\T{\char'136}}
        \Middle{\Empty}
        \Lower{\N{selector}}}
\Fil\Fil\EndDef

\Define{Selector}\Fil
\Alternatives
       {\Upper{\period\\\N{field identifier}}
        \Lower{\lbrak\\\RBW\comma\expression\\\rbrak}}
\Fil\Fil\EndDef
} % end of Expression

\vfil\eject

\Define{Constant}\Fil
\TwoAlts[{\sign\\\TwoAlts[\N{constant identifier}|\N{unsigned number}]}
        |\T{\singlequote}\\\Rept{\character}\\\T{\singlequote}]
\Fil\Fil\EndDef

\Define{Unsigned constant}\Fil
\Alternatives
       {\Middle{\N{constant identifier}}
        \Lower{\N{unsigned number}}
        \Lower{\KW{nil}}
        \Lower{\T{\singlequote}\\\Rept{\character}\\\T{\singlequote}}}
\Fil\Fil\EndDef

\Define{Unsigned number}\Fil
\digits\\\Optional{\period\\\digits}\\\Optional{\T{E}\\\sign\-\digits}
\Fil\Fil\EndDef

\Define{Unsigned integer}\Fil
\digits
\Fil\Fil\EndDef

\Define{Identifer}\Fil
\N{letter}\\
\Repeat{\Upper{\N{letter}}
        \Middle{}
        \Lower{\N{digit}}}
\Fil\Fil\EndDef
}
\vfil\eject
%%%%% End of PascalSyntax.TEX
```

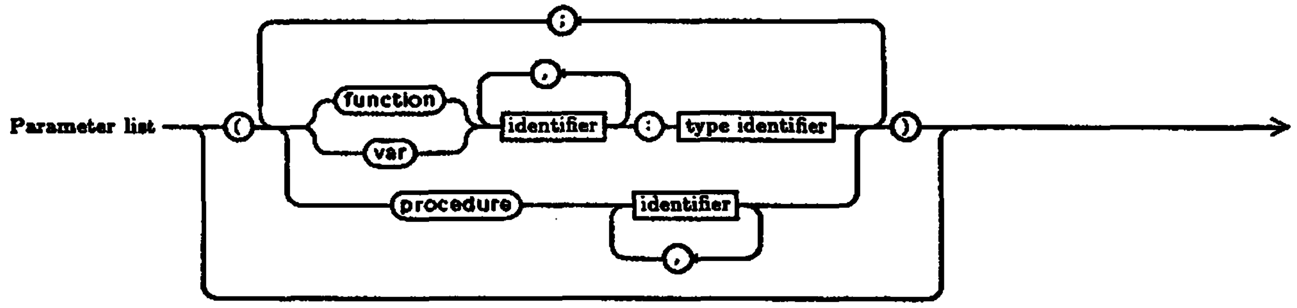


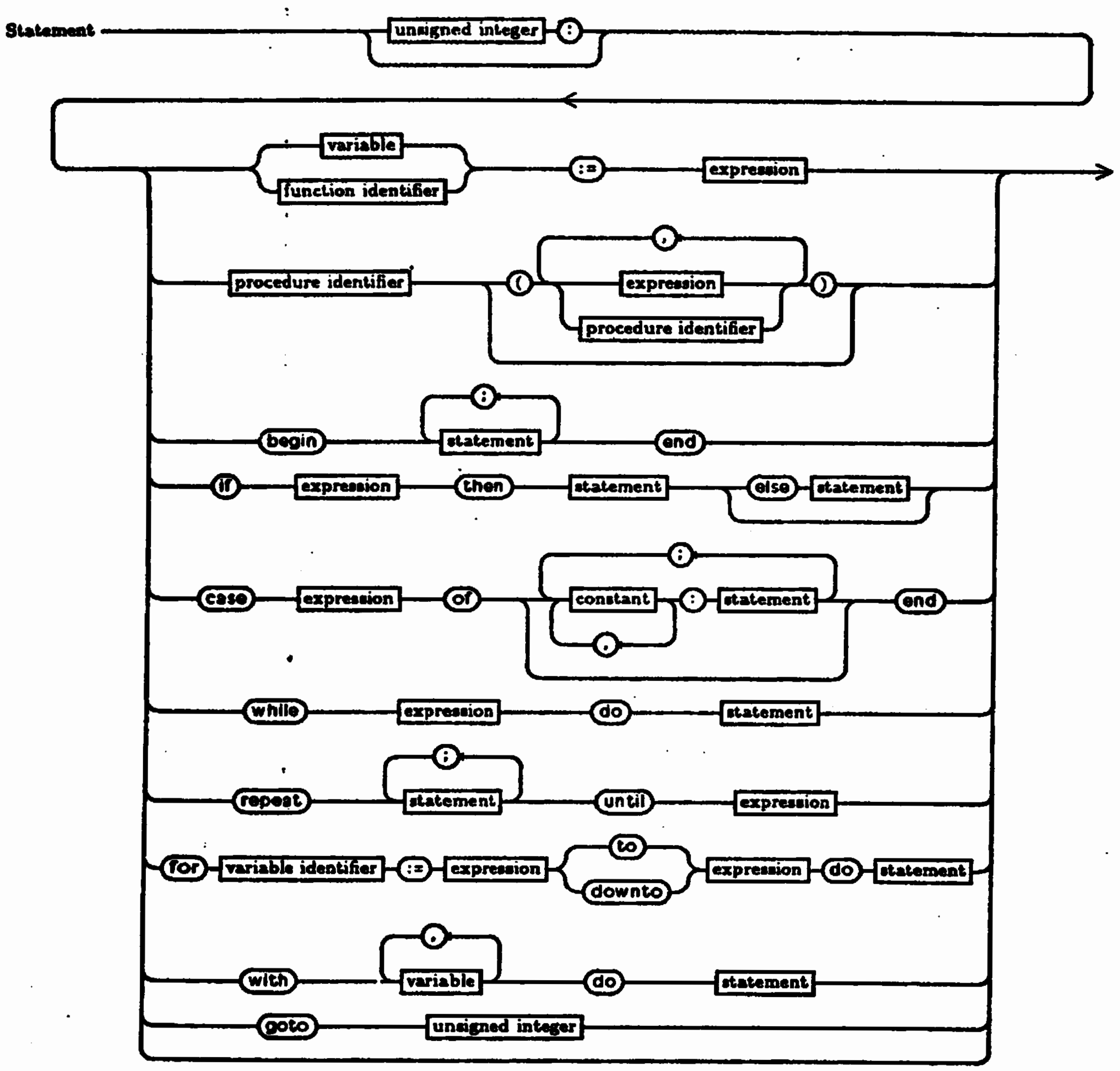
Editor's note: As you may have noticed on the first two pages of this paper, there are some problems with the font used to draw the quarter-circles. The paper copy which Mike Plass sent in with his tape looks quite a bit better, and the remainder of his diagrams are reproduced from that copy, which was prepared on one of Xerox's printers.

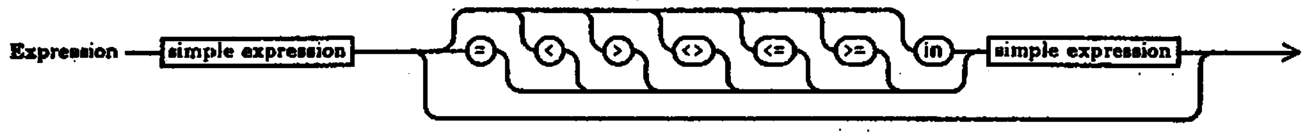

Type
simple type
↑
type identifier
,
packed
array
[
simple type
]
of
type
file
of
type
set
of
simple type
record
field list
end

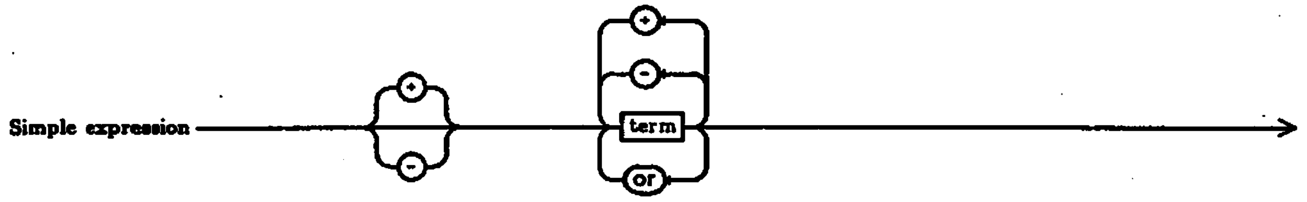

Simple type
type identifier
,
(
identifier
)
constant
..
constant

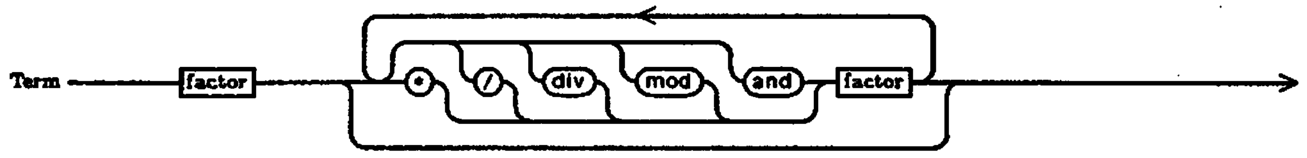

Field list
,
identifier
:
type
;
case
identifier
:
type identifier
of
,
constant
:
(
field list
)
;

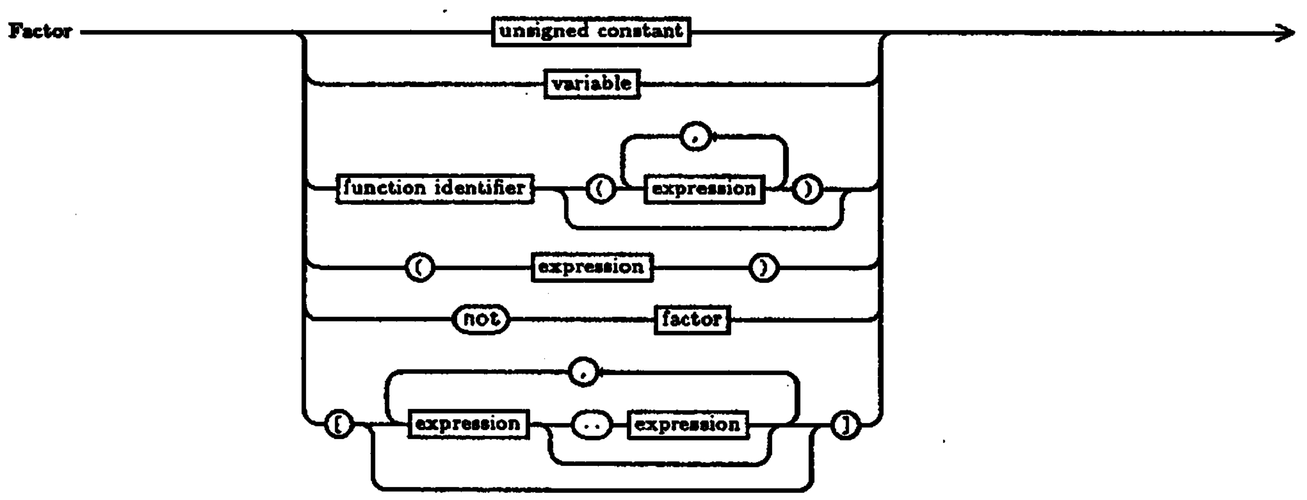

Parameter list
(
;
function
var
,
identifier
:
type identifier
)
procedure
identifier
,

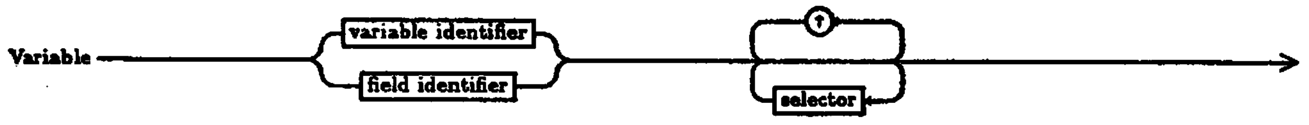

Statement
unsigned integer
:
variable
function identifier
:=
expression
procedure identifier
(
,
expression
procedure identifier
)
begin
;
statement
end
if
expression
then
statement
else
statement
case
expression
of
;
constant
,
:
statement
end
while
expression
do
statement
repeat
;
statement
until
expression
for
variable identifier
:=
expression
to
downto
expression
do
statement
with
,
variable
do
statement
goto
unsigned integer

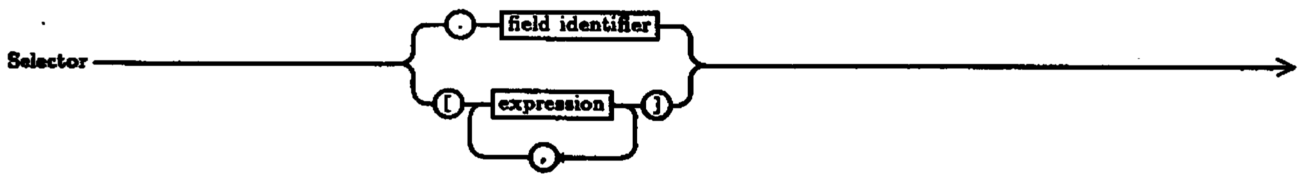

Expression
simple expression
=
<
>
<>
<=
>=
in
simple expression

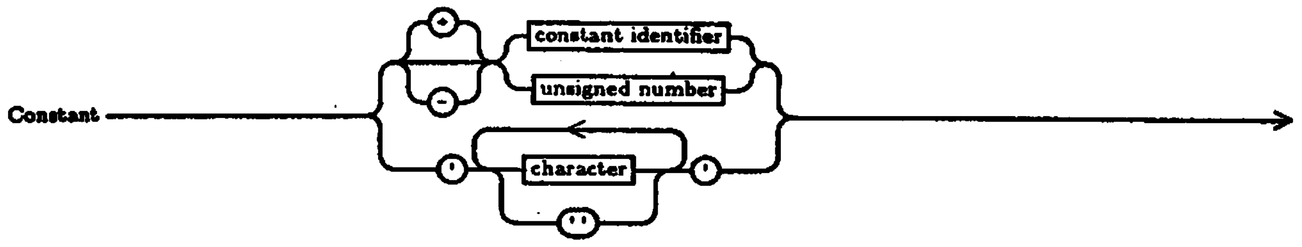

Simple expression
+
-
+
-
term
or

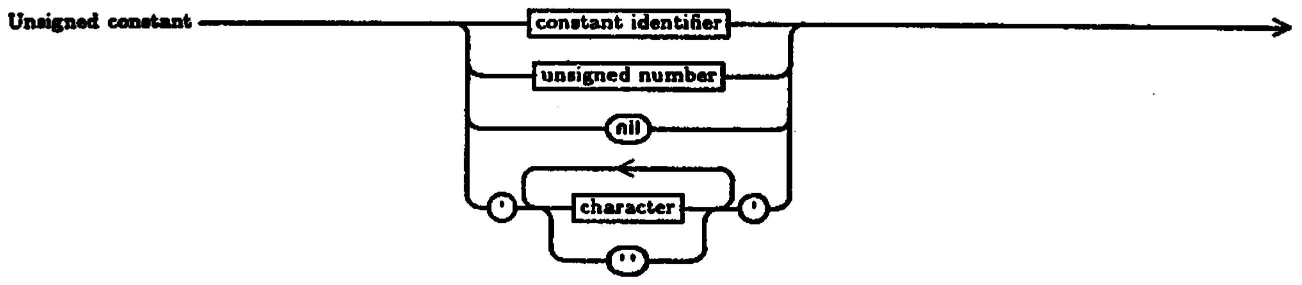

Term
factor
*
/
div
mod
and
factor

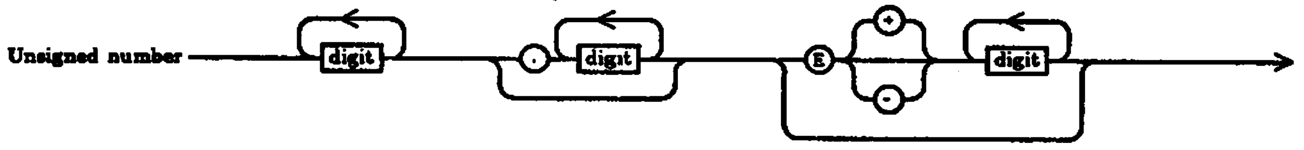

Factor
unsigned constant
variable
function identifier
(
,
expression
)
(
expression
)
not
factor
[
,
expression
..
expression
]

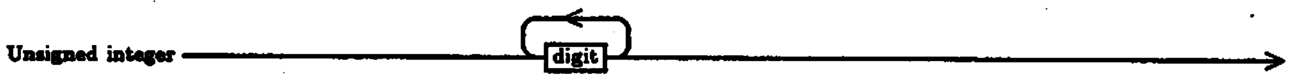

Variable
variable identifier
field identifier
↑
selector

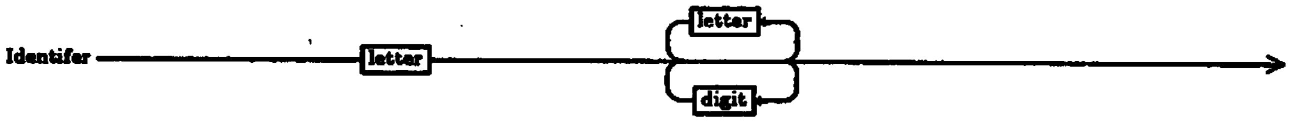

Selector
.
field identifier
[
expression
]
,


Constant
+
-
constant identifier
unsigned number
'
character
''
'


Unsigned constant
constant identifier
unsigned number
nil
'
character
''
'


Unsigned number
digit
.
digit
E
+
-
digit


Unsigned integer
digit


Identifer
letter
letter
digit

CHEMICAL NOTATION USING TEX

Monte Nichols
Barbara Beeton

The article below, from a mineralogy handbook, illustrates the fact that publishing in disciplines other than mathematics and computer science can benefit from the use of TEX. A few changes were necessary to fit the copy into TUGboat's narrow columns: the bar chart was truncated at 7.5cm (10cm in the original), and some line breaks and spacing were indicated manually where the appearance of the copy would otherwise have been less than desirable (the original column width was 3.5in, the TUGboat column width, 18.75pc = 3.125in, just enough of a difference to be troublesome).

The tables are built with ordinary `\halign` coding; especially attractive is the centering of headings over multiple columns in the table following *Powd. Pat.* The most interesting control sequences defined for this document are for the chemical formulas and the bar chart:

Chemical formula:

```
\def \F#1 {\if A\whatptsize
      {{\mathsy uzz\mathit adf ${#1}$}}
   \else{\if 9\whatptsize
      {{\mathsy vzz\mathit bef ${#1}$}}
   \else{\if 8\whatptsize
      {{\mathsy wzz\mathit cef ${#1}$}}
   \else{}}}}
```

This macro depends on the definition within the `\...point` macros (cf. `basic.tex`) of a control sequence `\whatptsize` which allows the current size to be tested; single-digit sizes are represented by one digit, and larger sizes follow the example of hexadecimal notation (`A` = 10, etc.). Notice that subscripts are defined to be the same size, regardless of level, and roman fonts are substituted for the usual math italic to simplify typing. Double braces ensure locality of these substitutions.

Bar chart:

```
\def\1{\hbox to 5mm
      {\hfill\vrule depth 3pt}}
\def\2{\hbox to 5mm
      {\hfill\vrule depth 6pt}}
\def\e#1 #2 {\hbox to #1mm
      {\hfill\vrule height #2pt}}
```

All dimensions in the original were `true`, but since the AMS version of TEX doesn't yet support that feature, they were replaced by ordinary dimensions.

The input looks like this:

```
\vbox{\hbox to 7.5cm{\vrule height 5pt
\e 10.09 4
\e 1.74 3
\e 4.25 9
\e 0.38 3
 . . .
} %end of hbox
\hrule
\hbox to 7.5cm{\vrule depth 8pt
\1\2\1\2\1\2\1\2\1\2\1\2\1\2\1
\hss} %end of hbox
\vskip 1pt
\hbox to 7.5cm{{\eightpoint
American Mineralogist {\bf 55}, 1100,
      (1970)}\hfill$2\theta\toright$}
} %end of vbox
```

The `\hss` following the `\1\2...` line is used to avoid an overfull box exactly the width of the last `\vrule`.

And finally, here is the article we've been talking about.

HEMIHEDRITE $ZnF_2[Pb_5(CrO_4)_3SiO_4]_2$

Morph. Triclinic–Pedial, 1; C_1

Habit. Well-formed doubly-terminated crystals from 0.2 to 10 mm in length. Elongated parallel to [001] with 80 forms reported. Twins most commonly by reflection in $\bar{2}\bar{2}3$ as penetrations of crystals of opposite hand to form an X, V, or Y shape with c inclined at B°. Less commonly by reflection in $0\bar{1}0$; also by reflection in $0\bar{1}2$.

Phys. $H = 3$; $\rho_{meas} = 6.42$; $\rho_{calc} = 6.50$; poor cleavage on {110}. Color bright orange to henna brown to almost black. Streak saffron yellow (Munsell 5Y8/10).

Struct.[2] The structure is similar to those of the tsumebite series and contains a Zn coordinated by four O and two F; the Pb environments are quite varied. Cr and Si are regularly four-coordinated by O.

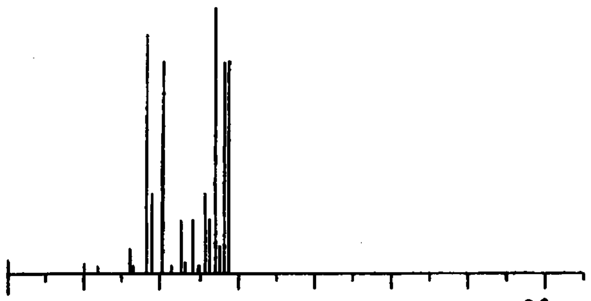
Occur. In a secondary oxide vein assemblage suggestive of alkaline solutions of relatively low Eh. Associated with cerussite, wulfenite, vauquelinite, willemite, and mimetite. Primary minerals include galena, pyrite, and tennantite.

Distr. Known only from two Arizona localities. The type locality is the Florence Lead-Silver Mine, Pinal County; also at the Rat Tail Claims, near Wickenberg, Maricopa County.

Name. For its distinct hemihedral morphology.

Ref.

1. Williams, S. A. and Anthony, J. W. (1970), Hemihedrite, A New Mineral From Arizona, *Am. Min.* **55**, 1088–1102.
2. McLean, W. J. and Anthony, J. W. (1970), The Crystal Structure of Hemihedrite, *Am. Min.* **55**, 1103–1114.



American Mineralogist 55, 1100, (1970) $2\theta \rightarrow$

Powd. Pat. Debye–Scherrer (114.6 mm; CuK_α; Visual I).

STRONGEST LINES		LARGEST d SPACINGS					
3.301	100	8.765	4	4.364	80	3.478	30
4.872	90	7.481	3	4.136	3	3.399	20
4.364	80	5.512	9	3.909	20	3.301	100
3.164	80	5.384	3	3.820	4	3.234	10
3.102	80	4.872	90	3.676	20	3.164	80
2.924	55	4.675	30	3.584	3	3.102	80

Struct. Cell. $P1 - C_1^1$; $Z = 1$;

$$a = 9.497(1), \quad b = 11.443(2), \quad c = 10.841(2)$$
$$\alpha = 120°30(1)', \quad \beta = 92°06(1)', \quad \gamma = 55°50(1)'$$
$$a : b : c = 0.830 : 1 : 0.947$$

Chem. Substitution of Zn for Pb noted in some samples.

	1	2
ZnO	2.7	3.93
PbO	73.0	70.5
Cr_2O_3	19.7	19.5
SiO_2	3.9	3.2
F	1.2	5.1
$-O \equiv F$	−0.5	−2.1
Total	100.0	100.0

1. $ZnF_2[Pb_5(CrO_4)_3SiO_4]_2$.
2. Average of several partial analyses.

Opt. Thin section shows feeble pleochroism with $Z > Y > X$. Relief extreme; dispersion resembles horizontal dispersion.

$\alpha = 2.105(5)$ yellow $(2V_z)_{meas} = 92°(-)$
$\beta = 2.32\ (2)$ yellow $(2V_z)_{calc} = 88°(+)$
$\gamma = 2.65\ (2)$ orange

* * * * * * * * * * *

Problems

* * * * * * * * * * *

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

Balancing Columns of Text and Translation

In the last issue, Johnny Stovall asked about a macro that could adjust the width of each column of two-column output so that the lengths of the two columns will be equal. His application involves typesetting original texts in parallel with translations. As long as reliable estimates of the relative length of the two segments are available, a simple technique can be used. `\varunit` can be set to the width available for text in both columns (excluding margins), and the actual width of each column can then be set in terms of a percentage of vu.

The following macro illustrates this approach using `\hbox par`:

```
\input basic
\varunit 6in % Space available for both
             % columns, exclusive of margins
\def\intercolumnspacing{\hskip .5in}
% Arguments to \trans are percentages
% of total width for first column,
% contents of first column, percentage
% of total width for second column, and
% contents of second column, respectively.
% The two numbers should sum to 100.
\def\trans#1#2#3#4{
  \hbox{\hbox par 0.#1vu
  {#2}\intercolumnspacing
  \hbox par 0.#3vu{#4}}
}
```

Here is a simple example:

```
Now is the time for all good men    Now is the time for all
to come to the aid of their party.  good men to come to
Now is the time for all good men    the aid of their party.
to come to the aid of their party.  Now is the time for all
Now is the time for all good men    good men to come to
to come to the aid of their party.  the aid of their party.
Now is the time for all good men    Now is the time for all
to come to the aid of their party.  good men to come to
Now is the time for all good men    the aid of their party.
to come to the aid of their party.  Now is the time for all
Now is the time for all good men    good men to come to
to come to the aid of their party.  the aid of their party.
```

And continue ...

Here, it is assumed that there is text extending across the full page width above or below the translations. This macro must be modified if the text segments include multiple paragraphs. A similar macro could be used to redefine \hsize if the entire text has this form.

Input-Dependent Macro Redefinition

Also in the last issue, Mike Spivak asked about a pair of macros \data and \list. Successive calls to \data would save text that could later be retrieved by calling \list, so that \list{1} would produce the parameter passed on the first call to \data, \list{2} would produce the parameter passed on the second call to \data, and so on.

Macros to solve this problem are fairly straightforward—\data can concatenate its successive arguments on a single string, separating them with some delimiter, and then \list can pick off the necessary text string. The concatenation of the passed strings, however, is created with \xdef and a problem arises when some of the arguments themselves contain \xdefs that the user does not want evaluated until \list is invoked.

A solution is possible using the technique illustrated by Patrick Milligan's \DefineFont macro published in TUGboat, Vol. 2, No. 2. The macro \macrolist is defined to contain a sequence of otherwise unused macro names (e.g., \aa, \bb, \cc ...). Each time \data is called, it removes the first item from \macrolist, saves it on a list \listitems, and then defines the specified macro to invoke the argument text. Thus, if \macrolist starts \aa\bb\cc..., the first call to \data sets \listitems to \aa, sets \macrolist to \bb\cc\dd..., and defines \aa to be whatever the argument to \data was. The \xdef to modify \listitems does not cause \aa to be expanded, because \listitems is redefined before \aa is defined. To retrieve the stored macros, \list must pick off the appropriate token from \listitems and execute it.

The algorithm just sketched is the heart of the macros shown here. The situation is somewhat more complicated, however. \list works by using \xdefs applied to \listitems. To prevent \list from expanding any \xdefs that might be contained in the argument text \data has so far received, \listitems is constructed with the undefined macros \a, \b, \c, etc. After \list has selected the appropriate item, \a is locally defined (i.e., defined within a group) to invoke \aa, \b to invoke \bb, and so on through the ugly macro \equivalences. The selected macro is then invoked within the group that defines these equivalences. To allow all this, \macrolist is really initialized to \aa\a\bb\b\cc\c\dd\d....

These macros also use the \Apply, \First, and \Rest macros used by Patrick in \DefineFont. \list uses the recursion macros Brendan McKay described in the same issue of TUGboat. The obvious restriction that these macros are limited to 26 calls to \data can, of course, be extended by reinitializing \macrolist. A less obvious restriction comes from the local equivalence of \a and \aa, etc. A macro defined with \def within an argument text is usable only from within the same argument, since the argument is invoked within a group. \gdef and \xdef, of course, have more permanent effects. This restriction can be surmounted by redefining \list to explicitly test its argument and to call \aa if the argument is 1, \bb if it is 2, etc. Unaesthetic as such an approach may be, it is not much worse than the current definition of \equivalences. Note that a "groupless if" such as McKay and Spivak also discuss in the last issue must be used to avoid the same problem.

```
\xdef\listitems{}

\def\data#1{
  \if !\macrolist!{
    \send9{Error: Maximum number of
                  data items exceeded}}
  \else{
    \Apply {\First} to {\macrolist!} ->
          {\macro} % Get macro name
    \Apply {\Rest} to {\macrolist!} ->
          {\macrolist} % Remove code from list
    \Apply {\First} to {\macrolist!} ->
          {\macroprime} % Get macro name
    \Apply {\Rest} to {\macrolist!} ->
          {\macrolist} % Remove code from list
    \xdef\listitems{\listitems\macroprime}
    \define\macro{#1}
  }
}

\def\define#1#2{
  \let \Gdef=\let
  \xdef\Define{\Gdef #1}
  \let \Gdef=\gdef
  \Define{#2}
  }

\def\list#1{{
  \xdef\temp{\listitems}
  \repeat #1\times
    \if !\temp! {\setcount9 0\gdef\macro{}}
    \else{\Apply {\First} to {\temp!} ->
                {\macro}
          \Apply {\Rest} to {\temp!} ->
                {\temp}}\endrepeat
  \equivalences
  \macro}}

% The macro \macrolist describes the set of
% macro names available to \list and \data.
% These macros should never be explicitly
% invoked by the user.
```

```
\def\macrolist{\aa\a\bb\b\cc\c\dd\d\ee\e
  \ff\f\gg\g\hh\h\ii\i\jj\j\kk\k\ll\l
  \mm\m\nn\n\oo\o\pp\p\qq\q\rr\r\ss\s
  \tt\t\uu\u\vv\v\ww\w\xx\x\yy\y\zz\z}

% \equivalences equates pairs of consecutive
% macros declared in \macrolist

\def\equivalences{\def\a{\aa}\def\b{\bb}\def
  \c{\cc}\def\d{\dd}\def\e{\ee}\def\f{\ff}\def
  \g{\gg}\def\h{\hh}\def\i{\ii}\def\j{\jj}\def
  \k{\kk}\def\l{\ll}\def\m{\mm}\def\n{\nn}\def
  \o{\oo}\def\p{\pp}\def\q{\qq}\def\r{\rr}\def
  \s{\ss}\def\t{\tt}\def\u{\uu}\def\v{\vv}\def
  \w{\ww}\def\x{\xx}\def\y{\yy}\def\z{\zz}}
```

A Macro That Prints Its Name

Anthony Siegman asks whether it is possible to devise a TEX macro that will process a one-word argument as both a word of text and a control sequence.

Suppose the argument is entered as {name}, or alternatively as {\name} (one or the other, not both). He wants a macro that will:

(a) Accomplish the result \xdef\name{\count 4}
(b) Put both the numerical value of \name and the word "name" into the output text.
(c) Send both the numerical value of \name and the word "name" to an external file.

Mike Spivak has written the following sequence

```
\def\findname{\chcode'134 12 \findit}
\def\findit #1{\finditt#1\endd}
\def\finditt #1#2\endd{\gdef
          \thename{#2}\chcode'134=0 }
```

which allows the input

```
\findname{\foo}
```

to define \thename to be "foo". However, despite appearances, \findname has no parameters; {\foo} is the text following \findname. This technique illustrates that TEX can be made to treat what appears to be a control sequence as two tokens: the text character \ followed by a word of text. It can not be extended to solve the complete problem presented by Siegman, since once TEX has determined the character class of a symbol, the character cannot be reinterpreted. Thus, any one occurrence of the four characters "\foo" can be interpreted either as four text characters or as one control sequence, but can not sequentially be interpreted both ways.

Nevertheless, macros approximating what has been asked for can be written. Instead of defining a macro \foo, the following \setname macro, given the argument "foo", allows another macro to return the value of \count4 at the time \setname{foo} was called. This new macro is invoked by a single character, @, so, in effect, the control sequence defined by \setname is @foo rather than \foo. Of course, "@foo" is not really a control sequence, but it may be used as such. Actually, "foo" is an argument to the macro @. @ expects its parameters to be terminated by a space, the user must remember to include such a space even in situations (e.g., before punctuation) where it would not be required for an actual control sequence.

The \setname macro defined below defines a list of its arguments on successive calls, separated by semicolons and a similar list of the values of \count4 each time it was called. The macro @ then looks through the list of names until it finds the one matching its argument and returns the corresponding \count4 value.

```
% Used to put a space between the number
% and name output by \send
\def\space{ }

% Initialize lists
\def\namelist{}
\def\numberlist{}

% Define a new "name"
\def\setname#1{
     \xdef\namelist{\namelist #1;}
     \xdef\numberlist{\numberlist\count4;}
     \xdef\number  % Number followed by
          {\count4 } % space #1; Put numerical
     \number\ #1  % value & word into text
     \send9{\number\space #1}% Send number
     }              % & name to external file

% Retrieve a previously saved value
\def\name#1 {\xdef\nametemp{\namelist}
             \def\numbertemp{\numberlist}
             \xdef\lookfor{#1}
             \gdef\action{\next}
             \next
            }

% Test if next name on temporary list is one
% needed. Note, if the name is not on the list,
% an error message will be generated since
% \nosuchname has not been defined
\def\next{\if !\nametemp!{\gdef\action
                 {}\nosuchname}\else{}
        \Apply {\first} to {\nametemp!} ->
                 {\nextname}
        \Apply {\first} to {\numbertemp!} ->
                 {\nextnumber}
        \Apply {\rest} to {\nametemp!} ->
                 {\nametemp}
        \Apply {\rest} to {\numbertemp!} ->
                 {\numbertemp}
        \stringeq{\nextname}{\lookfor}
        \if T\stringeqv {\nextnumber
                 \gdef\action{}}\else{}
        \action
       }

\def\first #1;#2!{#1}
```

```
%       Returns first token from list
\def\rest #1;#2!{#2}
%       Returns list with first token removed

% Allow \name to be invoked by the
% single character @
\def\ {\name}
\chcode'100=13
```

Testing the Equivalence of Strings

To test the equivalence of strings, the macro \stringeq is called. Shown below, this macro uses the "groupless if" techiques described independently by McKay and Spivak in Vol. 2, No. 2.

```
% String equivalence--evaluates to T
% if first argument is same string
% as second, F otherwise

% \stringeq returns its answer by
% setting \stringeqv
\def\true{\gdef\stringeqv{T}}
\def\false{\gdef\stringeqv{F}}

% Main "routine"  Save arguments in \stringa
% and \stringb and start testing
\def\stringeq#1#2{\xdef\stringa{#1}
  \xdef\stringb{#2}
  \gdef\endap{\enda}
  \gdef\endbp{\endb}
  \gdef\compcharp{\compchar}
  \enda
  }

% Test if at end of first string
\def\enda{\if !\stringa!{\gdef\endbp
                 {\nulltest}}\else{}\endbp}

% Test if at end of second string
\def\endb{\if !\stringb!{\false
        \gdef\compcharp{}}\else{}\compcharp}

% Compare next characters
\def\compchar{\Apply {\First} to {\stringa!} ->
 {\firsta}\Apply {\First} to {\stringb!} ->
 {\firstb}\if \firsta\firstb
{\Apply {\Rest} to {\stringa!} -> {\stringa
}\Apply {\Rest} to {\stringb!} -> {\stringb}}
\else{\false\gdef\endap{}}
\endap}

% When at end of first string,
% test if also at end of second
\def\nulltest{\if !\stringb!{\true}\else
                            {\false}}
```

* * * * * * * * * * *

PROBLEMS FROM THE TEXARCANA CLASS: ANSWERS, AND ANOTHER PROBLEM

These problems are from the videotaped TEXarcana Class taught by Don Knuth last March. The solutions given here are based on the ones distributed in class. Some parts of the solutions may depend on features which, although installed in the current version of TEX at Stanford, may not yet have been implemented elsewhere; an attempt has been made to note such features.

Problem no. 1:

Type:

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

\vskip 2pt
\noindent Architecture Critic
```

To get:

–Allan Temko
Architecture Critic

```
\def \hide#1{\hbox to 0pt{\hss #1}}
```

Problem no: 2:

Type:

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

To get:

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

```
\font H=cmr10 at 30pt
\def \fancy#1{{\:H\save0\hbox{\chop to 0pt{\hbox{\lower 12pt
     \hbox{#1}\hskip .1em}}}\hangindent 1wd0 for 2\noindent
     \hide\box0\hskip -.1em}}
```

The font specified here requires that magnification be implemented; it is also possible to use `cmr30`, if that is available. (At the AMS, neither is available for the Alphatype, on which this page was prepared, and the large "S" has been obtained from another source and pasted in.)

Note the use of `\hide` in the last line of this solution; alternatively, this line might begin "`\hskip -1wd0 ...`".

Problem no. 3:

Type:

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

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

To get:

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

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

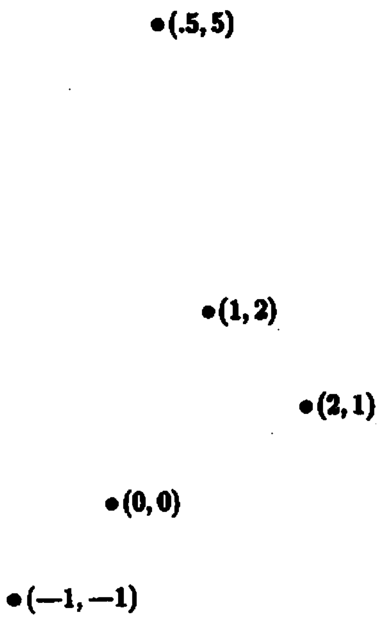
```
\def \signed#1#2{\parfillskip 0pt{\unskip\penalty 1000\hfil\penalty 200
\hskip 2em\hbox{}\penalty 1000\hfil{\sl#1\/} (#2)\par}}
```

Problem no. 4:

Type:

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

To get:



```
\lineskip 0pt
\baselineskip 0pt
\varunit .5in
\topspace 6vu \def \point#1#2 {\vbox to 0pt
    {\vss\vbox to #2vu{
     \hbox{\hskip 2vu\hskip #1vu
      $\bullet\,(#1,#2)$}\vss}}}
```

Problem no. 5:

Type:

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

To get:

End of a paragraph.

<div align="right">This is the first line.
This is the second line.
The third.
The last.</div>

Beginning of another paragraph.

```
\def \rightjustifythefollowing{\par
   \chcode'15=12\penalty1000
   \vskip-12pt\let\rjn=\rj\rj}
\chcode'15=12
\def \rj#1
{\rjustline{#1}\rjn}\chcode'15=5 %
\def \endrightjustify{\gdef\rjn{\endrj}}
\def \endrj{\chcode'15=5\penalty1000\vskip-12pt}
```

Problem no. 6:

Type:

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

To get:

How do you do this?

```
     L
     o          .
     o          e
     k          n
                o
     a
     t      stranger

     t          s
     h          i
     i          h
     s          t

  strange       t
                a
     p
     i          d
     l          n
     e          A
     .
```

```
\chcode`44=12
\def\linedown#1{\gdef\ans{}\Linedown#1$}
\def\Linedown#1{\if $#1{\gdef\next{\ans}}
    \else{\xdef\ans{\ans#1\cr}\gdef\next{\Linedown}}
    \next}
\def\lineup#1{\gdef\ans{}\Lineup#1$}
\def\Lineup#1{\if $#1{\gdef\next{\ans}}
    \else{\xdef\ans{#1\cr\ans}\gdef\next{\Lineup}}
    \next}
\chcode`44=3
```

How do you do it faster?

```
$$\vcenter{\halign{\hfil#\hfil\cr
\linedn{Look at this {strange}pile.}}}$$

\def\linedn#1{\gdef\ans{}\Linedn#1$}
\def\Linedn#1{\if $#1{\gdef\next{\ans}}
    \else{\gdef\nans{\#1\cr}\gdef\next{\ans\Lndn}}
    \next}
\def\Lndn#1{\if $#1{\gdef\next{\nans}}
    \else{\gdef\ans{\#1\cr}\gdef\next{\nans\Linedn}}
    \next}
```

On the next page appears the "Challenge problem" presented to the TEXArcana attendees. The solution will appear in the next issue.

Challenge Problem

First page of output:

•If I have all the eloquence of men or of angels, but speak without love, I am 1
simply a gong booming or a cymbal clashing. •If I have the gift of prophecy, 2
understanding all the mysteries there are, and knowing everything, and if I have
faith in all its fulness, to move mountains, but without love, then I am nothing at
all. •If I give away all that I possess, piece by piece, and if I even let them take 3
my body to burn it, but am without love, it will do me no good whatever.

•Love is always patient and kind; it is never jealous; love is never boastful or 4
conceited; •it is never rude or selfish; it does not take offence, and is not resentful. 5
•Love takes no pleasure in other people's sins but delights in the truth; •it is always 6 7
ready to excuse, to trust, to hope, and to endure whatever comes.

•Love does not come to an end. But if there are gifts of prophecy, the time 8
will come when they must fail; or the gift of languages, it will not continue for

Second page of output:

ever; and knowledge—for this, too, the time will come when it must fail. •For our 9
knowledge is imperfect and our prophesying is imperfect; •but once perfection 10
comes, all imperfect things will disappear. •When I was a child, I used to talk 11
like a child, and think like a child, and argue like a child, but now I am a man,
all childish ways are put behind me. •Now we are seeing a dim reflection in a 12
mirror; but then we shall be seeing face to face. The knowledge that I have now is
imperfect; but then I shall know as fully as I am known.

•In short, there are three things that last: faith, hope and love; and the greatest 13
of these is love.

You should use this text file ⟶

```
3 Mar 1981    21:86          LOVE.TEX[  1,DEK]              PAGE 2-1

@ If I have all the eloquence of men or of
        angels, but speak without love, I am simply a gong booming
        or a cymbal clashing.
@ If I have the gift of prophecy, understanding all the mysteries
        there are, and knowing everything, and if I have faith in all
        fulness, to move mountains, but without love,
                nothing at all.
          @ For our what I possess, piece by piece, and if I even
          @ but once perfe   burn it, but am without love, it will
          @ When I was a child,
              child, and argue like a
                ways are put behind me.
          @ Now we are seeing a dim reflectio ealous;
                be seeing face to face. The knowleo
                imperfect; but then I shall know as fully

          @ In short, there are three things that last: faith, hop-
                love; and the greatest of these is love.

          \flushpage
```

The driver file says:

```
\input basic
\def\pagesize{1.8 in}
<secret code to do
this formatting>
\setcount1 1  % starting verse number
\input love[1,dek]
\end
```

* * * * * * * * * * *

Letters et alia

* * * * * * * * * * *

To the Editor:

I wonder how much serious interest there is amongst TUGboat readers in radically simplifying and speeding up the capture and presentation of complex mathematical text. I have designed an unexpected 'mutation' of the standard Qwerty keyboard which, in mock-up form, allowed—with generous allowance for making and correcting mistakes—simulating the capture of a standard book page full of equations with double levels sub/super-scripts, fractions, differentials and integrals, etc., in a few seconds over twelve minutes. Commercial typesetters estimate at least thirty minutes would be needed even on the most sophisticated equipment presently available. I have also found new and unexpected principles for displaying several typesizes and typefaces in an immediately and exactly recognisable way (without using graphics) that help increase error awareness. The speed of the keyboard comes from its physical design, rather than other 'tricks'; people quite inexperienced in typesetting have remarked how easy it is to use and set complex work that would have completely puzzled them beforehand—it is good for use by relatively unskilled people.

I would like to set up production for several kinds of units if there is sufficient interest of a serious nature. However, until the advent of TEX the worldwide interest in special maths terminals has been minuscule; it is now uncertain, but probably greater. That is why I would like to hear of interest from TUGboat readers.

There are several useful units in mind. An "Idiot Keyboard" for those well versed in TEX, that merely generates editable code and/or TEX code as a direct output; a "Semi-Intelligent Keyboard", basically an 'Idiot Keyboard' with a video output and able to interact with a host; an "Intelligent Keyboard", with a monitor and two floppy discs, able to read editable code and generate TEX code from that, and assistance facilities (e.g., spelling as well), also interacting with a host; a "Realistic Display Maths Terminal", having the performance of an 'Intelligent Keyboard', but giving realistic display of maths, that is also fully editable on screen (no coding shown) and showing typefaces and sizes in an immediately recognisable way, also assistance facilities and interaction with a host; a "T&M Host" also having a "Realistic Position Maths Display" (graphics) and a proofing printer. A 'Realistic Position Maths Display' terminal would allow any maths office typist (or perhaps any other) easily to set the most complex material and automatically generate and edit TEX code, for example.

I would very much like to hear from people who have a real interest in such units and/or systems. The nature of the response to this enquiry will very much determine whether there is significant interest. Items would be expected to give substantial savings in salaries of dedicated programmers, and compare well with currently available equipment.

J. M. Cole
17 St Mary's Mount
Leyburn
North Yorkshire DL8 5JB, England

* * * * * * * * * * *

FORMATTING A BOOK WITH TEX: EXPERIENCES AND OBSERVATIONS

Michael Sannella
MIT Laboratory for Computer Science

In January of 1980 I was hired by Professors Harold Abelson and Andrea diSessa of MIT to format a book they had written: *Turtle Geometry, The Computer as a Medium for Exploring Mathematics*. They had used a computer to write and edit the book—the text was stored on-line—and they wanted me to use TEX to produce the final formatted copy. In part, they hoped that using a computer formatter would be cheaper than traditional typesetting. However, they were also interested in the experiment of publishing a computer-formatted book, and curious about the ways that computer formatting systems could change the relationship between authors and publishers, giving an author more control over a book.

When I was hired, I didn't know anything about TEX, or about book publishing. In the process of formatting the book (which took more than a year) I learned a lot about TEX, about the problems associated with book-quality formatting, and about the interaction of computer formatters with the world of book publishing. This article is an attempt to record my experiences formatting *Turtle Geometry* and some thoughts I have had concerning computer formatters in general. I hope that information will be useful to other people who are trying to format large documents, such as books, using computer formatters. I will try to talk about general problems I encountered rather than about the details of how I fixed each problem. Also, I will try to be as simple as possible—you won't need to know anything about TEX to read this article.

The first part of this article deals with the experiences I had formatting *Turtle Geometry*. Interacting with the people at MIT Press, learning TEX, and using TEX to format the book all presented problems. Many problems stemmed from the fact that I was using TEX, an extremely complex formatting program. However, all of my problems couldn't be blamed on TEX. *Turtle Geometry* was a very large, very complex document, with innumerable difficult figures, equations, etc. It would have been difficult to format this book using any method. Eventually, I dealt with all of these difficulties, and produced an extremely good-looking book. (*Turtle Geometry, The Computer as a Medium for Exploring Mathematics*, MIT Press, June 1981)

Since I finished formatting *Turtle Geometry* I have been thinking about computer text formatters. In the second part of this article, I discuss some of my ideas on the design of text formatting systems. My experiences formatting *Turtle Geometry* give me an interesting perspective for looking at this area. First, I talk about the problem of designing formatting languages which give the user the correct level of control over the formatting process. Then, I discuss the problem of integrating high-level aesthetic formatting goals into a computer formatting system. I don't claim to have any great answers to these problems—I just want to point out some problems that future text formatters will have to deal with.

I would appreciate receiving any comments that anyone may have concerning this article. I can be contacted be sending U.S. mail to:

Michael Sannella
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Or (for those of you with access to the ARPAnet) by sending computer mail to MJS at net site MIT-AI.

Formatting *Turtle Geometry*

I worked on this project part-time from January, 1980, through March, 1981. My job was defined very simply: to produce a camera-ready copy of the book, using TEX. Aside from that, nothing was specified. I was confident that this goal was achievable, since Knuth had designed TEX specifically to format his books. However, I knew that I had a lot to learn about producing book-quality copy. During the fifteen months I worked on this project, I had a good opportunity to observe the interaction between the two worlds of computer text formatting and book publishing. Computer formatters are being used increasingly for large commercial formatting projects, so it is necessary to investigate how they can be used most effectively. I hope that this record of my experiences formatting *Turtle Geometry* will be useful to those people actively investigating this area.

Dealing With MIT Press

One of the first things I did after being hired to format *Turtle Geometry* was to talk with the people at MIT Press (the publishers of the book) and hear their views on the project. The book editor and the book designer were not computer scientists—they were mainly interested in producing a book. However, they were intrigued by the idea of using a computer formatting system, instead of traditional typesetting, and were willing to participate in the experiment of formatting the whole book by computer. They made a few things clear: first, the quality of *Turtle Geometry* must not suffer as a result of doing the formatting by computer. In the past, a few people have published books (mainly computer science textbooks) formatted using primitive text formatters and low quality printers, which looked horrible, and were impossible to read. The MIT Press was not willing to publish a book of such low quality. I was expected to produce computer-generated output fine enough so that, when it was photographed and printed, the book it would be virtually indistinguishable from a book typeset using the normal methods. They were not willing to sacrifice quality for the sake of our experiment. A second condition was that they did not want to put more work into this book then they would have if it was typeset normally. There would be little reason to use the computer if it took more work on their part. Within these restrictions, however, they were willing to be flexible. They were interested in what we could do using a computer formatting system.

The book designer gave me a set of specifications, written on a standard form. This included such information as the page dimensions, the placement of headings and page numbers, the fonts to be used with different types of text, etc. These specifications had been chosen so as to make the book as beautiful and readable as possible. Book designers use their experience with book formatting, and a long-cultivated sense of aesthetics, to choose a combination of design specifications such that the total effect is pleasing to the eye, as well as appropriate for the type of book being published. This is the art of book designing.

Turtle Geometry is a textbook dealing with mathematics, physics, and computer science. It contains mathematical equations, computer program listings, and figures, interspersed with paragraphs of text. The book designer decided how each of these objects should be formatted, so that the book would be readable, and would have the right "style." Each of these objects seemed to have fairly simple formatting—there were no fancy footnotes or other objects requiring complicated formatting—so I didn't anticipate any problems with the formatting of individual objects. However, the book as a whole was sure to present some problems, because there were literally hundreds of figures and equations that had to be formatted. I hoped that using a computer formatting system would allow me to cope with this problem better. In retrospect, formatting individual equations and figures didn't present many problems. The hard problems arose when I had to deal with the formatting of the book as a whole. In part this is because TEX can handle low-level details beautifully, but cannot deal directly with higher-level formatting concepts. I will discuss this problem more thoroughly later.

LEARNING TEX

After talking with the book designer, I had to learn about TEX. I knew that Donald Knuth at Stanford University had developed TEX in order to format his series of books, *The Art of Computer Programming*. Supposedly, TEX was a book-quality typesetting system particularly optimized for formatting mathematics. So I picked up a copy of Knuth's TEX manual and proceeded to read it.

My first impression was that TEX was very complicated. Knuth's manual tries to present the language "gently," a little at a time, so that the reader is not overwhelmed, but it only succeeds in being confusing. This style only served to make it difficult to use the manual as a quick reference. Eventually, I wound up reading the manual cover-to-cover three times, until I had a good idea about how the whole TEX system worked.

TEX can be used to produce very high-quality formatted output on a high-resolution printer. However, this comes at a price. The TEX user has to specify exactly how he wants to format his document using a complicated formatting language. TEX is an improvement over many other formatters in that it uses some of the vocabulary and concepts of printing and typesetting, but it still requires the user to specify the formatting in excruciating detail. In general, I would characterize TEX by saying that its "output" is good, but its "input" is bad. It is possible to produce beautifully-formatted copy, but the specifications necessary to achieve this are very complex.

Rather than explicitly specifying the exact formatting for every object in a document, TEX provides a macro facility which allows one to encapsulate and name a sequence of commonly-used formatting specifications, and to refer to them by using the macro name. TEX's macros are implemented using simple text substitution—when you reference a macro, TEX acts as if the macro text were substituted for the macro reference. Macros can be defined to take arguments, which are inserted into the text of the macro in the same way that the macro is inserted into the text file. It was obvious that I should use macros to specify all of the different objects that would appear in *Turtle Geometry*. I began creating a file of macros. Initially, they were very simple, and I was able to get them working very quickly. I gradually refined them, changing the macro definitions until they fulfilled the specifications for the book.

I had a lot of problems using macros because of the way that they are implemented in TEX. In most cases, simple text-substitution worked well, but sometimes I wished that it were possible to write "smarter" macros. One problem was that the exact effect of a set of formatting commands depends on where they occur. A TEX macro has no way of detecting where it has been placed, so you can get very strange results by putting a macro in the wrong place. A similar problem occurred because the arguments to a macro are blindly inserted into the macro text, and there is no way to check whether they are appropriate. If the arguments are not of the correct form, the macro may do something quite unexpected. (I give an example of this second problem below, in the section on formatting figures.) Both of these problems happen as a result of people making mistakes—using macros incorrectly—so it could be claimed that this is not a deficiency of TEX. However, this situation influences the way that people use TEX. In order to use a macro correctly, you need to know exactly where it can be used, and the precise form of all of its arguments. Therefore, all macros need to be supplied with extensive documentation concerning their use. This is the same documentation problem that people have to deal with when maintaining programs written in computer languages, raised to a higher degree. Most computer languages allow programs to be somewhat isolated from each other, with only minimal communication between them—this allows one to consider the effects of one program apart from the others. Macros in TEX get much of their meaning from the context within which they are inserted. A macro is meaningless by itself. Consequently, this makes the tough problem of documentation even tougher. While I was developing the macros for *Turtle Geometry*, I found that I had to write extensive notes to remind myself exactly where and how to use my macros. In spite of this, I made a lot of mistakes which were difficult to discover and to fix. Luckily, the macros would not have to be used by anyone else. It would not have been easy to write macros that anyone could use. In general, I would not want anyone else to use my macros Unless they understood them completely, it would just be too easy to make a mistake. Because of this problem, I predict that it will be difficult to set up "libraries" of commonly-used TEX macros, in an effort to prevent duplication of effort. Instead, most people using TEX will have to start from scratch, and write their own macros.

One unfortunate characteristic of TEX is that it is difficult to debug your formatting specifications if there is some type of error. TEX has a large set of error messages, but these only signal when you specify formatting illegally. Most of the time, TEX formats your document without complaining, and there is something strange in the output. What can you do? Usually you can figure out where something went wrong, but sometimes this is very difficult. This problem is compounded by the use of macros, which may do unpredictable things if you use them incorrectly. This makes TEX very hard to use.

FORMATTING THE TEXT OF *Turtle Geometry*

Once I learned how to use TEX and developed the formatting macros, I began to edit the text of the book, inserting the macro invocations where appropriate. The book was first written and printed using a primitive text formatter called TJ6, so the text files (one per chapter) contained text interspersed with TJ6 commands. All of the TJ6 commands had to be removed, and TEX commands had to be inserted. I eventually deleted all of the TJ6 commands using a text editor, then scanned the straight text which remained, and inserted TEX commands as needed.

Most of the time formatting the text was no problem—the job was tedious, but not difficult. Occasionally I came across an object that had to be formatted somewhat strangely, and I had to do a lot of thinking and experimentation to make TEX do what I wanted. However, with most objects all I had to do was to insert the appropriate macro invocations, and TEX would format them correctly.

Using TEX's macro facility to specify formatting turned out to be very useful when I was not exactly sure how something should be formatted. For example, *Turtle Geometry* contained many vector equations. The authors and the book designer were not exactly sure how they wanted to represent vector variables. So I defined a macro called `\vect`, and used it to specify all vector variables. Then, I printed out a section of the book several times, defining `\vect` a different way each time. By changing that single macro definition, I could represent all vectors in an entire section by letters with arrows on top of them, or by boldface characters. The only way to judge which notation was better was to see how it looked in print. Using macros allowed me to experiment with very little effort.

FORMATTING FIGURES

Of all of the objects in *Turtle Geometry*, the hardest to format were the figures. MIT Press and the authors had decided that the figures would be drawn by a technical illustrator, and pasted onto the camera-ready copy of the formatted document. Therefore, spaces had to be left for the drawings. Formatting a figure proved more complicated than simply leaving a specified amount of space on a page, however. The book designer had specified that a single figure should contain a caption, and an arbitrary number of subfigures, each with an optional label, organized one or two or three in a row across the page. I had to develop macros to allow me to construct complex figure boxes.

One problem I ran into was determining exactly how much space to leave for a particular figure. If I put in too much space, the drawing would look strange surrounded by emptiness. If I didn't leave enough space, on the other hand, it would be impossible to paste the drawing into the final copy of the document. Given a drawing, I measured it, and give its height as an argument to a figure-generating macro. In most cases, this produced a good-looking figure, with just enough blank space to paste in the appropriate drawing. However, there were a fair number of figures which didn't have labels for the subfigures. Because of the way that TEX macros are implemented, there is no way for a macro to detect when it is given a null argument. When given a null label argument, my figure formatting macros still allocated space for a label, which resulted in a figure with too much space. Eventually, I solved this problem by subtracting an appropriate amount from the size I gave as an argument to the figure macro, to compensate for the extra space added for the non-existent label. This was a rather inelegant solution. In retrospect, it would have been better for me to have defined separate macros for figures with labels and figures without labels. The best solution, if TEX had had the capability, would have been to define a figure macro that could detect when it is given a null label, and format the figure accordingly. TEX could be greatly improved by extending its macro facility so that a macro could test its arguments.

Page Breaking and High-Level Formatting

As I have mentioned before, I didn't encounter many problems with formatting most of the individual objects in the book. To format a figure, or a program listing, or an equation, I just had to use the appropriate macro from the set I developed. Admittedly, constructing and debugging that set of macros was a slow and painful process, but I only had to do that once. The real problems began after I had formatted the individual objects and was trying to bring them all together. The book designer at MIT Press had given me certain aesthetic rules to follow when formatting pages. Unfortunately, TEX was not designed to deal with such high-level formatting concepts. As an example, consider the placement of figures within a document. Each figure is referred to at a specific point within the text. The readability of a document is improved if figures are placed so as to minimize the amount of page-flipping a reader has to go through to associate references with figures. For this reason, the book designer at MIT Press specified that a figure should be placed at the top of the page containing its reference, if possible. If that was not possible, the next best choice would be to have the figure on the facing page, to the left or to the right. If that too was not possible, then the figure should be on the next possible page later on in the document. Unfortunately, TEX cannot deal with figure placement in these terms. For one thing, there is no notion of "facing pages" in TEX. This is a serious failing, since commercial publishers design document formatting in terms of how each "spread" (facing left and right pages) looks. More generally, TEX does not give the user very much control over which page a figure is placed upon.

Another formatting rule I was given was that the pages should all be of the same length, if possible, but a page could be a line short or a line long, if that would help the formatting of the page. In TEX, the height of a page is set to a specific distance using a `\vsize` command. Page breaks are generated to fit as closely as possible to that limit, without going over. There is no provision in TEX for changing this number dynamically, on a page by page basis, subservient to other formatting needs. Within TEX all formatting decisions have equal weight, which sometimes unnecessarily restricts the possibilities for formatting something, and reduces TEX's power.

If I only had to deal with a small, simple document, the aesthetic goals mentioned above would not be very important. It is not difficult to place a figure in a small document consisting of a few pages, and TEX would probably position it correctly, using its simple figure-placement algorithms. However, *Turtle Geometry* was a very large document, with many figures and equations. When you have a large number of figures in a document, placing these figures becomes a much more complicated problem, and it is harder for TEX to place them correctly. It can't deal with the aesthetic concepts that come into play with large documents. The first time I formatted *Turtle Geometry* using TEX, it formatted the individual figures and equations correctly but the total effect was horrendous. The book was unreadable—of much lower quality than you would get using traditional human typesetting.

I had to find some way to correct this situation. First, I tried coercing TEX into doing the correct page formatting by adding formatting commands at places where it broke pages and placed figures incorrectly. This proved to be very difficult. The problem was that I could easily prevent TEX from breaking a page at a particular bad place, but it was harder to ensure that it would break it at a good place. I was never sure of the effect of adding any particular formatting command until I saw the text reformatted. Coercing TEX involved endless experimentation, just to get around the inadequacies of TEX's page formatting algorithm. I realized quickly that this method was unsuitable. I was spending my time fighting TEX, trying to merge its page formatting with my aesthetics, rather than worrying about the formatting of the book.

Rather than having both TEX and myself handle the page breaking and figure placement, I decided to simplify things by doing all of the page breaking by myself. To be exact, I inserted an `\eject` command every place that I wanted a page break, and specified to TEX that it shouldn't do any page breaks itself. So that I would not have to figure out by myself where all of the page breaks should go, I developed a macro called `\bookmark` which I inserted into each file containing a chapter of the book. The `\bookmark` macro told TEX that all page breaks before the point where it occurs will be handled by explicit `\eject` commands, but that TEX should attempt to generate page breaks from that point on. I eventually developed a routine for handling the page breaks in a chapter: I would insert a few `\eject` commands, move the `\bookmark` macro call forward to the end of my `\ejects`, and run the file through TEX. Seeing how TEX would format the next few pages helped me decide where I wanted to insert page breaks. Often, I didn't have to do anything more than insert `\eject` macros where TEX had broken the page before, but most of the time I wanted to move figures around, and change the exact placement of the page breaks. It was necessary to do this work a few pages at a time, because each formatting change I made could send effects percolating through the rest of the document in unpredictable ways.

By doing the page breaking and figure placement myself, I was able to optimize the page formatting with respect to aesthetic rules that I could not express with TEX. Consequently, I produced a book of very high quality.

LOOKING BACK

Formatting *Turtle Geometry* was a long, difficult job. Everything did not go as smoothly as it may seem from what I have said above. One problem was that the macro development, the figure formatting, and the editing of the actual text of the book were all happening at the same time. It was very difficult to coordinate these separate activities. Another problem, which is sure to occur whenever a document is available on a computer system, was that of "freezing" the text of the document. When it is easy to change the text, authors are very reluctant to finish editing. There is always a great temptation to fix one more error. At some point, however, it is necessary to decide that the text will not change any more.

A typical example of the organizational problems I had to deal with was that associated with putting the figures in the book. It took a long time to get the figures drawn and checked. Until the drawings were in final form, I couldn't measure them and give these measurements to the figure macros. Therefore I couldn't do the page formatting, since the sizes of the figures critically affected the location of page breaks. To cope with this situation, I worked on each chapter separately, so that at any one time different chapters could be at different stages of production—the authors might still be editing the text of chapter 5 while I was sizing the figures for chapter 1, and formatting its pages.

I made many mistakes, and had to do some things several times. I was learning what to do, and developing new methods for doing things, while I was formatting the book. Everything would have been much better if all of the people working on *Turtle Geometry* had sat down before the work was started, and developed a routine for producing the book. We could have avoided a lot of problems, and produced a better book with less effort, if we had been more organized.

Thoughts on Text Formatters

Current computer text formatting systems suffer from many deficiencies, which become increasingly apparent as people attempt to use them in more and more situations. Some of these problems are due to formatting languages that cannot cope with the aesthetic ideas that book designers have developed from long experience with document production. Other problems result because formatters are designed with small documents in mind, and cannot deal with the formatting concerns of large documents. I don't claim to have "the solution" for all of these formatting problems. However, I have come up with a few ideas about how text formatters could be improved.

FORMATTING LANGUAGES AND CONTROL

Computer-controlled printers deal with documents at a very low level. To a graphics-oriented printer, a document is a set of commands which specify the exact positions of individual dots, lines, and characters on a page. Obviously, people do not want to have to specify the formatting of their documents in such excruciating detail. Instead, they use a computer text formatter, which could be considered as a translator between an abstract, high-level formatting language and the specific detailed language of a printer. A formatter allows its user to specify document formatting in terms which are more abstract and general, such as words and paragraphs and lines of text. Using these specifications, the formatter decides where each character of the document should be placed on each page.

An important characteristic of a formatting language is the level of detail with which the user has to specify the formatting of a document. Suppose that a formatting language deals only with high-level formatting concepts, and the user wishes to specify the formatting of a particular object a little more explicitly. This could happen if the formatter doesn't format something exactly right automatically, and the user wants to fix this. Another possibility is that the user may wish to create a one-time special-case object, which needs to be formatted strangely, and the user is willing to take the extra time to specify it in detail. (As an example of this, in *Turtle Geometry* there were only two or three tables, and it was easier to specify them in detail, rather than developing general macros for formatting tables.) In this case, the only solution is for the user to try to coerce the system into formatting correctly. This may involve tricking the formatter, making it treat some objects like other objects, and trying to second-guess the formatting algorithm. This is very difficult to do, it requires an intimate knowledge of the internal workings of the formatting system which not all users may have, and it is extremely inelegant. The problem is really that the way such formatters are designed, there are simply some documents which cannot be produced. In the interest of making a formatter easier to use, the set of possible documents that could have been formatted with it has been constrained.

Now, let us consider the opposite extreme, where you have a very low-level formatting language, and you wish to do high-level formatting. In other words, you don't care exactly how your document looks, as long as it conforms to certain high-level formatting goals, such as having lines filled to a constant length, etc. Using a low-level formatting language, it should be possible to produce a document conforming to these high-level constraints, but you will be required to specify a lot of low-level details that you are not really interested in. Aside from the inconvenience of specifying the formatting in this much detail, this makes things very difficult if you wish to change your document later. The more detailed your specifications, the more work has to be done in order to change them. When you work with specifications at a low level of detail, you essentially have to calculate the formatting yourself, rather than letting a computer do it.

There is another, rather subtle, problem associated with formatting languages which use descriptions at a lower level than you need for a particular document. The greater the detail in which you specify the formatting of a document, the more you constrain the possible ways that it could be formatted. If you have to use a low-level language, you wind up making a lot of fairly arbitrary decisions about how something should be formatted at a low level, since you have to specify something. However, all of the formatting specifications are heeded by a computer formatting system, whether they are important to the user, or whether they were just specified arbitrarily. The more constrained a formatter is, the smaller the space of possible ways a document could be formatted, and the more likely that there will be no formatting which is acceptable. When a user is specifying how his document should be formatted, he should use the least possible amount of detail, while still mentioning those things important to him. In this way, the possibility is increased that the formatter can find an acceptable way to format his document.

Early text formatters were designed around the idea that documents were fairly simple, consisting of text organized into paragraphs, with an occasional heading or footnote. These programs would take the user's specifications, and format a document using rules internal to the formatter. It was possible to modify exactly how the formatting was done by specifying the values of certain parameters, but the formatting was essentially done by one "smart" program. These early formatters present a good example of formatting languages which work at too high a level for many formatting tasks. As long as the shape of your document conformed to the ideas explicit in the formatter's abstract paragraphs and words, it would be easy to to format your document. However, if your document contained something out of the ordinary that the program was not equipped to handle, there was little that you could do. No matter how much effort you were willing to expend to specify the detail, these early formatting languages wouldn't allow you to. The user simply had very little control over the formatting process. The shape of a document had to be specified in terms of the objects the formatter was built to deal with, then the user would just have to "throw it to the winds," and hope the formatter produced something acceptable.

TEX is similar to these early formatters, in that it deals with abstract ideas such as paragraphs and pages, and the user only has indirect control over how these objects are formatted. However, TEX also can deal with the idea of "boxes and glue," which allow a user to specify an object at a very low, but manageable, level. This facility is very useful for handling "special case" objects, such as complex figures, which would be too complicated for TEX to deal with as built-in objects. In essence, TEX allows the user to deal with formatting at two distinct levels of detail, or control. Either you can "throw your formatting to the wind" and let TEX take care of it all, or you can specify the formatting of the object in low-level detail. This is an improvement over the earlier text formatters, which gave you no choice, but it still proves inadequate in many situations. The problem is, a user sometimes want to have control over the formatting, without having to specify *all* of the formatting explicitly. For example, consider the problem I had while formatting the Turtle Geometry book with breaking pages, and placing figures. When I let TEX do all of the formatting by itself, it would do it incorrectly, in part because it doesn't consider some of the aesthetic considerations I am concerned with. For a while, I tried tricking TEX's formatting algorithms into doing the correct thing, but this involved going through all sorts of complicated contortions. Finally, I was forced to deal with the problem at a very low level, by specifying all of the page breaks and figure placements myself, explicitly. I was able to get what I wanted, but it required a lot of work on my part. Another issue in this situation was that once I had done the page breaking by hand, that essentially "froze" that part of the document. If any changes had needed to be made to the document after I had done the page breaks, it would have been necessary for me to do all of that work over again. It would have been better if I had the ability in TEX to specify formatting at a higher level, while still retaining control over the formatting.

People who design text formatting systems need to worry about this issue. One way that formatters could be improved is by giving more control to the user, at different levels of detail, so that it is not necessary to specify document formatting at more than the minimum amount of specificity. Looking at TEX in particular, one way to improve it would be to introduce "smarter" boxes—objects that are not quite as explicit and low-level as boxes are in TEX. For example, you could have an object called a "paragraph box," which contains text that should be formatted as a paragraph. Eventually, this may not be formatted as a single rectangular box—the paragraph may eventually extend across several pages. Without explicitly specifying the exact formatting of every character in the paragraph, you could express parameters appropriate to the level of abstraction of a paragraph, such as line length, degree of raggedness, whether the paragraph can be broken, etc. To format any particular object in a document, simply express it in terms of a formatter object which deals with the correct level of abstraction.

One problem with implementing this scheme is that it could lead to a complicated formatting language with an unmanageably large number of objects defined. A way to deal with this is to only define a few objects, which can be used to specify formatting at differing levels of detail. For example, consider a "box" object that acts in the following way: given a set of objects as arguments, it formats them like a paragraph, using default width, raggedness, etc. values. If these values are given as parameters to the box, it will use them instead. You can imagine a whole set of parameters and defaults set up so that a single box object could be used for formatting a paragraph (at the most abstract level) or used exactly like a box in TEX (if everything, including vertical and horizontal sizes, are specified). A single object, with many parameters and defaults, could be used at many different levels of abstraction.

HIGH-LEVEL FORMATTING GOALS

One problem I have mentioned regarding current text formatters is that they do not deal with some of the aesthetic rules that book designers have developed after years of experience with formatted documents. I had to explicitly specify page breaking for *Turtle Geometry*, because TEX could not deal with some of these concepts. Let us examine this problem more closely. When breaking pages by hand, I had to keep in mind several aesthetic considerations. First, figures had to be positioned to be as near to their references in the text as possible (on the same page, or on the facing page, or on the next page). Second, I didn't want a page break in the middle of a computer program listing, if at all possible. Third, the length of each page should be nearly constant, plus or minus a line. In certain special situations, I had to worry about other aesthetic considerations, such as not having an equation directly under a figure, but this didn't come up too often. Mainly, I was concerned with choosing page breaks, and positioning figures, so as to produce a document optimized with respect to those three aesthetic rules given above.

Certain problems arise when dealing with high-level aesthetic formatting goals. It is difficult to explicitly specify such goals in a form that a computer can work with. Consider what a *human* has to go through to format a document while considering these goals. It is seldom the case, given a tricky formatting situation with a lot of figures, that all of these formatting goals are compatible. Often, they are contradictory. For example, suppose you have a page with a computer program listing in the middle, and a figure reference near the top of the page. Suppose that if you place the figure at the top of the page, this pushes the text on the page down far enough so that the computer program has to be broken between this page and the next one. Here you have a conflict between the rule that wants figures on the same page as their references and the rule that doesn't want computer program listings to be broken. Exactly what I would do depends on the exact situation. If the page in question was a left-hand one, it might be possible to move the figure to the facing right-hand page. Alternately, if the place where the program was to be broken was only a few lines into the program, I might just break that page short. (notice that in this second case I may be breaking the third aesthetic rule.) Making an aesthetic decision of this sort involves finding a compromise between several contradictory goals.

The problem of formatting a document with respect to certain aesthetic formatting goals can be reduced to the problem of considering two situations, where a part of a document is formatted in two different ways, and deciding which is "better." Usually there are only a few possibilities to consider in any formatting problem, and a computer could afford to try the various possibilities, and pick the "best" one. The key to this problem is obviously determining what the word "better" means with respect to a set of high-level aesthetic formatting goals. It is instructive to examine a mechanism that TEX has for dealing with this problem on a lower level. In order to optimally generate line breaks and page breaks within a paragraph, TEX associates an integer with each paragraph known as its "badness." Every line break generates a certain "penalty," another integer, which indicates how "bad" that break is. Line breaks have large penalties associated with them if the break requires hyphenating a word, or if a line is too long, among other things. Page breaks in a paragraph have large penalties associated with them if they generate widows (the first or last line in the paragraph alone on a separate page). The badness of a paragraph is calculated as the sum of all of the penalties in the paragraph. The formatting algorithms inside TEX are designed to make line and page breaks so as to minimize the total badness of a paragraph. This is a fairly flexible system—users can specify penalties explicitly at certain points to indicate places that are particularly good or bad to break at—and it formats individual paragraphs very well. However, this type of system would not be suitable for dealing with more complex formatting constraints. "Badness" is a simple, one-dimensional "score" which can be easily calculated and used for comparisons, but it is not powerful enough to express some of the more complex combinations of higher-level formatting goals. Consider the aesthetic goals given above, dealing with figure placement and computer program listings. You can't simply take a badness score with respect to one goal, and a badness score with respect to another, and sum them to get total badness. This is like adding apples and oranges. Aesthetic formatting goals combine in more complex ways than that. In a particular situation, it might be possible to represent a situation in terms of carefully-adjusted penalties, but this wouldn't work in general. You need a more complex, multidimensional, way of comparing two situations.

The fundamental problem with implementing high-level abstract formatting goals in a text formatting system is the problem of achieving a goal. Supposing that you have a way of unambiguously stating fairly complex aesthetic rules, how is the computer to make sure that they are achieved? Exactly what can the formatter do if the formatting it chooses disobeys one of the aesthetic rules? Consider the low-level decisions that a formatter makes when formatting a document. The decisions about exactly which word to break a line at, or exactly how much to stretch a line, are fairly arbitrary. Unless the user specifically specifies it, the formatter has a fair amount of leeway as to exactly how the document is formatted. Ideally, the low-level arbitrary decisions could be influenced by the high-level aesthetic rules. I admit that I don't know how this could be accomplished, but this is obviously what text formatters should work towards. As an intermediate step, formatting systems should be designed that can express the concepts of high-level aesthetic goals. Even if they couldn't use these goals to influence the formatting process, it would be worthwhile simply to flag violations of these rules, pointing out to the user places in the document where one or another of the rules are broken. This by itself would lead to better formatters, producing better-looking documents.

Contents

November 1981

***** Continued *****

Contents — Continued