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 P from Vrije University, Amsterdam.

* * * * * * * * * * *

DIABOLIC TeX

Timothy Murphy
Trinity College Dublin

Preamble

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

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

This brief account of our own efforts may t fore not be out of place. At the very lea may shame some of the TeXperts who have developed Diablo drivers to share their secrets us beginners.

The Diablo as printer

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

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

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

(3) All characters and symbols might be ma out of dots, using the graphics mode on the Di In effect this would make the Diablo analogous digitalised type-setter, albeit one of very low re tion.

Our calculations seemed to show that the solution would be impracticably time-consum each page taking more than half-an-hour to put We hope to implement the first solution sh

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 hori tal and vertical movements are treated in much same way; so only one of these is detailed.

Our PASCAL compiler PASC20 allows the in sion of header files containing CONST and T declarations. This useful feature greatly red the risk of incompatible modifications being n to different modules. Our header file TEXDIA. listed in Appendix C after the driver BYTDIA.

Our only real design decision was to accumu movements. TEX puts out a large number of re dant movements, e.g. successive DVI instruct might order an upward movement of 2 points, lowed by a downward movement of 10 points. prevent the Diablo from doing a St Vitus da we accumulate all movements until printing is minent. Thus a record is kept of the point (re realV) on the page where the "cursor" actuall as well as the point (H, V) where it should be, i movements to date had been implemented.

The actual position is only updated- making the appropriate horizontal and vert movements—when a print instruction is receive

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|\h0o\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
Department at Stanford. Professor Knuth h
working version of the UNDOC macro proc
written in its own language (DOC). UNDOC
piles itself into a Pascal program, thus UNI
is now available in Pascal. DOC is being use
the source language for new versions of TEXI
and TEX82. All three programs (both DOC
Pascal sources) are expected to be available for
ing to new machines in early 1982. TEX82
complete rewrite of TEX based on the experi
gained from Ignacio Zabala's translation of
TEX. Portability has been improved by removin
floating point operations. Another sticky portal
problem with the current Pascal TEX is initia
tion. Recall that installing a new TEX involves
ning the program TEXPRE, which makes a
file (called TEXINI.TBL) that represents the in
state of TEX's data structures (about 36K wor
size). On TOPS20, we then run TEX, which
in TEXINI.TBL, at which point we interrupt
process and save the current core image. When
users ask for "TEX", they get a copy of this
image, which continues execution from where w
terrupted the first TEX run. Thus, our users
saved the not-insignificant overhead of data s
ture initialization. The resulting core image is
smaller and faster than if the initialization
tions of TEXPRE were to be incorporated into
Unfortunately, we have found that the facilit
"saving an interrupted job's core image for later
tinuation" is not available in many environm
including VAX VMS, UNIX, and IBM timesha
systems. Consequently, TEX users outside of
DEC 36-bit world have TEX re-read TEXINI.
each time it is run, which is a significant
handicap. To help rectify the situation, TE
data structures will change to require less initia
tion. We also plan to make a program avai
that can read TEXINI.TBL and produce Pa
language initialization code to be inserted int
TEX Pascal source before compiling. Unfortuna
variable initialization is not standard Pasca
there must be different versions of this progra
the Hedrick compiler, Pascal/VS, VMS Pascal,