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

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

. . . . . .

Appendix: Comparison between version 0 and version 1.

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

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

Some ideas for version 2.

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

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

* * * * * * * * * * *

SOME FEEDBACK FROM PTEX INSTALLATIONS

Ignacio Zabala

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

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

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

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

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

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

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

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

Two more particularly interesting problems are worth mentioning here.

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

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

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

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

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

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

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

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

Tab characters not allowed in source.

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

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

No standard MAX and MIN functions.

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

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

U. OF MINNESOTA: CD CYBER (MIKE FRISCH)

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

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

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

MULTICS: BENSON MARGULIES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

instead of

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

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

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

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

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

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

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

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

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

TEX AND HYPHENATION

Frank M. Liang

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

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

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