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.

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

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

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

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

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

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

Time magazine algorithm

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

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

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

Patterns

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

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

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