The Ins and Outs of Pascal/3000 I/O

Susan R. Kimura
Hewlett-Packard

Numerous questions usually arise when a user first encounters I/O in Pascal. Some of these questions are:

- Do I have to use GET and PUT to do my I/O?
- What is the deferred GET all about?
- How exactly does EOLN work?
- How do READ and WRITE work?
- Why do READ and WRITE take longer than MPE's FREAD and FWRITE?

This paper addresses these questions. The discussion focuses on the standard Pascal file type called textfile. The first part discusses textfiles in general. The second part discusses the use of READs and WRITEs of textfiles. Some comparison with MPE's FREADs and FWRITEs is also made.

TEXTFILES

Pascal defines a file type as a structure which consists of a sequence of components of the same type. The following discussion focuses on a Pascal standard file type called TEXT. A variable of type TEXT is called a textfile. A textfile is a logical file which consists of components of type CHAR and is structured into lines separated by line markers. It is accessed sequentially. A textfile also has associated with it a buffer variable and a current component. The buffer variable is denoted f^ for a textfile f. It may be previewed prior to a read operation or modified prior to a write operation.

There are two primitives, GET(f) and PUT(f), which operate on a textfile. These primitives are used in conjunction with the buffer variable f^ to perform input and output. GET(f) is used for input and PUT(f) for output. The GET operation, as defined by Jensen and Wirth [3], advances the current file position to the next component and moves that component into the buffer variable. The PUT operation writes the contents of the buffer variable to the current component and advances to the next component.

The procedures RESET(f) and REWRITE(f) are provided to open a file for sequential I/O. RESET(f) opens the file f for read-only access. The file is positioned at the first component and a GET is performed. REWRITE(f) opens the file f for write-only access. The file is positioned at the first component. The file buffer variable f^ is undefined.

Note that this definition of RESET may cause problems for input from a terminal. If a GET, as defined above, is performed on the RESET a physical read from the terminal must be done in order to fill the buffer variable. The program is then paused for input from the terminal before the user has requested an input operation. The deferred GET, as defined in the HP Standard [2] and used by Pascal/3000, is a method which provides a solution to this problem. With the deferred GET the buffer variable is not filled on a GET. However, on the following reference to the buffer variable the current component is moved to the buffer variable and the file position is advanced to the next component. In other words, after a RESET the buffer variable f^ is undefined until an operation which references the buffer variable is requested. At that time the first component is moved to the buffer variable and the file position is advanced. An operation which would fill the buffer variable would be a reference to the buffer variable f^, or a call to READ(f,v), EOF(f), or EOLN(f).

EOF(f) and EOLN(f) are functions which indicate when the end-of-file has been

reached and when the end-of-line has been reached, respectively. EOF(f) is true when there are no more components to be read from file f. EOLN(f) is true when the current position is beyond the last component of a line. At this point the buffer variable f^ contains a blank.

An overview of the Pascal/3000 internals associated with a file would be useful at this point. When a file is declared, a block of storage is allocated which is used to keep track of the state of the file. This block of storage is called the file control block and contains information such as a readable flag, a writeable flag, an end-of-line flag, an end-of-file flag, a get flag, the current record length, the MPE file number, the file buffer variable, and the current position. Another block of storage is allocated which buffers the input or output of a physical I/O operation. This block of storage is called the file buffer. When an I/O operation is requested, the Pascal/3000 run-time library accesses and updates the information in the file control block. In the case of an input operation a record may be read from the file into the file buffer and characters moved from the file buffer to the buffer variable. In the case of an output operation characters may be moved from the buffer variable to the file buffer and the contents of the file buffer written to the file.

Consider the following program which uses GET(f) for reading characters from a textfile f. The physical file associated with f contains only one line with the sequence 'AB'.

```
        PROGRAM pascal (f);
        VAR
           f : TEXT;
           c : CHAR;
           b : BOOLEAN;
        BEGIN
 (1)    RESET(f);
 (2)  c := f^;
 (3)  GET(f);
 (4)  c := f^;
 (5)  GET(f);
 (6)  b := EOLN(f);
 (7)  GET(f);
 (8)  b := EOF(f);
      END.
```

The results after the execution of each statement are:

	c	b	feof	feoln	fcpos	fchar	fgetok
(1)	?	?	false	true	0	?	true
(2)	'A'	?	false	false	1	'A'	false
(3)	'A'	?	false	false	1	'A'	true
(4)	'B'	?	false	false	2	'B'	false
(5)	'B'	?	false	false	2	'B'	true
(6)	'B'	true	false	true	2	' '	false
(7)	'B'	true	false	true	2	' '	true
(8)	'B'	true	true	true	2	' '	false

Feof, feoln, fcpos, fchar and fgetok are variables in the file control block. Feof is the end-of-file flag, feoln the end-of-line flag, fcpos the current character position index, fchar the buffer variable and fgetok the get flag which indicates that the buffer variable may be updated.

On the RESET(f) in (1), the textfile f is opened for read-access. Fcpos is set to the first position, which is 0. Fgetok is set to true but fchar is still undefined. When the buffer variable is referenced in (2) an FREAD is done, filling the file buffer with the line 'AB'. Fchar is filled with the first character, 'A', and fcpos is updated. When GET(f) is called in (3), fgetok is set to true, but fchar is not updated. Statements (4) and (5) are similar to statements (2) and (3). When EOLN(f) called in in (6), the fcpos is beyond the last character. Consequently, fchar is filled with a blank as defined in Pascal and feoln is set to true. The GET(f) in (7) is used to get past the end-of-line marker by setting fgetok to true. Finally, the call to EOF(f) in (8) triggers another FREAD which returns an end-of-file condition and feof is set to true.

Writing to a file using PUT(f) is a straightforward case. Consider the following program which writes to the textfile f:

```
        PROGRAM pascal (f);
        VAR
           f : TEXT;
        BEGIN
 (1)    REWRITE(f);
 (2)    f^ := 'a';
 (3)    PUT(f);
 (4)    f^ := 'b';
 (5)    PUT(f);
        END.
```

The results after the execution of each statement are:

	fcpos	fchar
(1)	0	?
(2)	0	'a'

```
(3)      1       'a'
(4)      1       'b'
(5)      2       'b'
```

On the REWRITE(f) in (1) the textfile f is opened for write-access. Fcpos is set to the first position. The buffer variable fchar is undefined until the literal 'a' is assigned to it in (2). On the PUT(f) in (3) the contents of fchar is moved to the file buffer and fcpos is updated. Statements (4) and (5) are similar to statements (2) and (3). When the end of the program is encountered, the file buffer is automatically flushed. An FWRITE is done to the file f which now contains the sequence 'ab'.

READ, READLN, WRITE, WRITELN

While GETs and PUTs may be used for textfile I/O, they are not the most efficient method. To facilitate textfile I/O the predefined procedures READ, READLN, WRITE, and WRITELN are provided.

The forms of the READ statement are:

```
READ(f,v);
READ(f,v1,...,vn);
READLN(f);
READLN(f,v);
READLN(f,v1,...,vn);
```

The forms of the WRITE statement are:

```
WRITE(f,e);
WRITE(f,e1,...,en);
WRITELN(f);
WRITELN(f,e);
WRITELN(f,e1,...,en);
```

These procedures allow the user to read variables and write expressions of type char, integer, real, longreal, boolean, user-defined enumeration, PAC, and string. 1 The f parameter may be omitted. In this case the standard Pascal textfiles INPUT and OUTPUT are used for READs and WRITEs, respectively. The parameter v1,...,vn indicates that an arbitrary number of variables may be read. Likewise, the parameter e1,...,en indicates that an arbitrary number of expressions may be written. They need not all be of the same type. Furthermore, when reading or writing integer, real, longreal or user-defined enumeration, internal conversions occur. These conversions, from ascii to binary representation in the case of a read, or binary to ascii representation in the case of a write, greatly simplify the work for the user.

On a READ or READLN a sequence of characters which conform to the syntax of the type of the variable are read and converted from ascii to binary, if necessary. After a READ the file position is set after the last character read on the current line. After a READLN, any remaining characters and the end-of-line marker in the current line are skipped over and the file position is set at the start of the following line.

Conversely, on a WRITE or WRITELN, an expression is converted from binary to ascii, if necessary, and written to the file buffer. After a WRITE the file position is set to the position after the last character written. On a WRITELN an end-of-line marker is written after the last character 2 and the file buffer is written to the file using an FWRITE.

The write statements also allow formatting of output by specifying a field width parameter m. For reals and longreals, a decimal place parameter n may also be specified. The forms of the write expression are:

```
e
e:m
e:m:n
```

If no formatting is specified, a default field width is used. If m is specified, the expression is written in the field width specified with right-justification. For reals and longreals, if n is specified, the expression is written in fixed-point format with n decimal digits.

Consider the simplest case of reading in a sequence of characters into a variable c of type CHAR with the following statements from a textfile f which contains two lines, 'AB' and 'DEF'.

```
          PROGRAM pascal (f);
          VAR
             f : TEXT;
             c : CHAR;
             b : BOOLEAN;
          BEGIN
(1)       RESET(f);
(2)       READ(f,c);
(3)       READ(f,c);
(4)       b := EOLN(f);
(5)       READ(f,c);
(6)       READ(f,c);
          END.
```

Because the characters are being read one at a time, this sequence of statements is equivalent to:

```
          PROGRAM pascal (f);
          VAR
             f : TEXT;
             c : CHAR;
             b : BOOLEAN;
          BEGIN
(1)       RESET(f);
(2)       c := f^; GET(f);
(3)       c := f^; GET(f);
(4)       b := EOLN(f);
```

```
(5)     c := f^; GET(f);
(6)     c := f^; GET(f);
        END.
```

The results after execution of each statement are:

	c	b	feoln	fcpos	fchar	fgetok
(1)	?	?	true	0	?	true
(2)	'A'	?	false	1	'A'	true
(3)	'B'	?	false	2	'B'	true
(4)	'B'	true	true	2	' '	false
(5)	' '	true	true	2	' '	true
(6)	'D'	true	false	1	'D'	true

The textfile f is opened for read-access on the RESET(f) in (1). Fcpos is set to the first position. Fgetok is set to true but fchar is still undefined. When READ(f,c) is called in (2), an FREAD is done, filling the file buffer with 'AB'. Fchar is filled with the first character, 'A', and fcpos is updated. The contents of fchar is then assigned to the variable c. In addition, because a GET(f) is part of the READ, fgetok is set to true. Statement (3) is similar to statement (2). When EOLN(f) is called in (4) fchar is filled with a blank because fcpos is beyond the last character. The next call to READ(f,c) in (5) the blank in fchar is assigned to the variable c and fgetok is set to true. Finally, on the READ(f,c) in (6), another FREAD is done, filling the file buffer with 'DEF'. Fcpos is initialized to the first position, fchar is filled with 'D' and its contents assigned to the variable c.

If the READ(f,c) in (5) is replaced by READLN(f) with no variable to read, this would be equivalent to a GET(f) and the corresponding result would be:

	c	b	feoln	fcpos	fchar	fgetok
(5)	'B'	true	true	0	' '	true

In this case, fcpos is set to the first position, fgetok is set to true and the variable c remains unchanged.

As stated above READ(f,v) allows the user to read not only characters but also integers, reals, enumerated types, PACs and strings. Internally, when reading a variable of type INTEGER from a textfile f, the following sequence of events occur in the Pascal run-time routine for reading an integer:

(1) A procedure is called to verify that the file is open. This procedure searches a linked list of opened files. If the file is not found on the list, an error is reported.

(2) The read-access flag in the file control block is checked to verify that the file has read-access. If it does not, an error is reported.

(3) A procedure is called to verify that an end-of-file condition does not exist. A physical read using FREAD may occur. If an end-of-file is encountered, an error is reported.

(4) The buffer variable is scanned to skip over initial blanks. Characters may be moved from the file buffer to the buffer variable. FREADs may occur if end-of-line markers are encountered.

(5) A procedure is called to verify that an end-of-file condition was not encountered while skipping blanks. If so, an error is reported.

(6) A string of digits is moved into an internal buffer until a non-digit or end-of-line marker is encountered. The digits are moved from the file buffer, to the buffer variable, and then to the internal buffer.

(7) The compiler library routine EXTIN' is invoked to do the conversion to binary.

An analogous sequence of events occur when reading a real or longreal variable with the exception that reading the string of digits is more complex due to the decimal point and scale factor which may be present.

This run-time routine shows how much overhead there is compared with doing an FREAD. The user who uses FREAD has direct control over the state of his file. Consequently, the checks for whether the file is open and has read-access are not necessary. Since the Pascal

run-time routine does not know the dynamic flow of a user program these checks are necessary. On the other hand, the user who uses FREAD is responsible for doing the ascii to binary conversion. This is automatically done by the run-time routine. The primary overhead in the Pascal run-time library, is the movement of characters from the file buffer, to the buffer variable, then to the internal buffer. These steps are done in order to maintain the file control block in the correct state at all times.

Now consider the case of writing an integer to a textfile f. The following events occur:

(1) A procedure is called to verify that the file is open. If it is not, an error is reported.

(2) The write-access flag in the file control block is checked to verify that the file has write-access. If it does not, an error is reported.

(3) A check is made that a valid field width has been requested. If the field width is not valid, an error is reported.

(4) The compiler library routine IN-EXT' is invoked to convert the binary representation to an ascii string.

(5) The correct field width to use is calculated. The user-specified field width may be over-ridden if the converted string does not fit within the user-specified width.

(6) A WRITELN is done if the converted string does not fit on the current line.

(7) A procedure is called to write out the converted string. If the converted string is shorter then the determined field width, an appropriate number of blanks are inserted so that the string will be right-justified within the determined field width. The characters are moved from the string to the file buffer and the current position index is updated.

As demonstrated above, there is also some overhead in writing an integer to a textfile. Checks must be done to verify that the file is in the proper state. The conversion of the integer from binary to ascii representation must be done. Finally, the ascii representation must be moved to the file buffer. For the user using FWRITE, the check that the file is opened for write-access is not necessary. However, the conversion from binary to ascii, as well as formatting of output, which are taken care of by the run-time routine, must be done by the user.

Finally, consider the case of reading a variable of type PAC from a textfile. The following events occur:

(1) A procedure is called to verify that the file is open. If it is not open, an error is reported.

(2) The read-access flag in the file control block is checked to verify that the file has read-access. If it does not, an error is reported.

(3) A procedure is called to check for an end-of-line condition. An FREAD may occur. If an end-of-file condition is encountered, an error is reported. If an end-of-line condition is encountered, the PAC variable is blank filled and control is returned to the user program.

(4) If an end-of-line condition was not encountered, characters are moved from the file buffer to the buffer variable, then to the PAC variable, one character at a time, until the variable has been filled or an end-of-line condition is encountered.

(5) If the PAC variable was not filled, the remaining character positions are blank-filled.

As with the other run-time routines, checks must be made to verify that the file is opened with the appropriate access. When checking for the end-of-line condition, a physical read using FREAD into the file buffer may occur. Procedures are called to move the characters from the file buffer into the buffer variable and then into the PAC variable. Finally, the variable is blank-filled if necessary. This description shows that reading a PAC variable, which may be thought of as similar to an FREAD of a PAC variable, is more complex. The primary overhead is in accessing the routines which maintain the state of the file control block which are not neccessary for user FREADs.

To summarize, a great deal of overhead is involved in accesssing textfiles because the state of the file control block must be maintained, especially with respect to the buffer variable and the current position. The fact that textfiles have end-of-line markers complicate reading and writing of textfiles. On the other hand, the use of textfile I/O eliminates the need for the user to use or write his own conversion routines when reading and writing

integer, real, longreal, boolean, and user-defined enumeration. Furthermore, formatting of output is done for the user using the write routines.

The author hopes that this discussion has eliminated some of the "mystery" associated with textfile I/O and that this information may aid in the user's deveopment of Pascal programs.

Footnotes

1 In Pascal/3000 PAC is used to denote the type PACKED ARRAY [1..N] OF CHAR. String is a predefined packed structured type consisting of components of type CHAR. It has a maximum length but its actual length may vary dynamically at run time.

2 On the HP3000 the end-of-line marker is a logical marker. There is no actual end-of-line marker written to the file buffer.

Bibliography

[1] American National Standard Pascal Computer Programming Language. (1983). ANSI/IEEE 770X3.97-1983.

[2] Hewlett-Packard Standard Pascal Report. (1983). Release 2.

[3] Jensen, Kathleen and Wirth, Niklaus. (1975). Pascal User Manual and Report. Springer-Verlag. New York.

[4] Pascal/3000 Reference Manual. (1981). Hewlett-Packard.

Acknowledgement

I would especially like to thank my colleague Chris Maitz for his expertise and advice. I would also like to thank my project manager, Jean Danver, and my colleages, Jon Henderson and Pat Miyamoto, for proof-reading this paper.

Biographical Sketch

Name : Susan R. Kimura

Title : Member of Technical Staff

Employer : Hewlett-Packard Computer Languages Lab

Job Responsibilities : Involved in current product engineering on the COBOLII and Pascal/3000 compilers since 1980.

Education: BA, University of Hawaii, Psychology MA, University of Hawaii, Japanese Linguistics MS, University of Wisconsin, Computer Sciences

Marital Status : Married with one child
